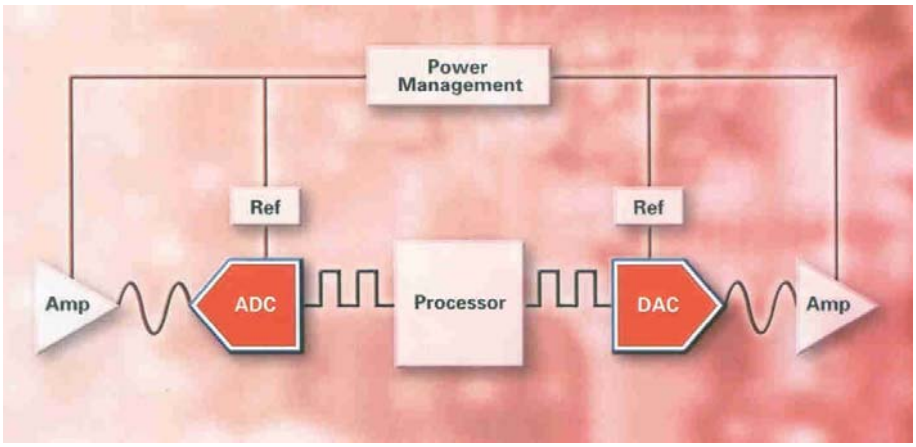
# Sistemi a microprocessore

**lunedì 19 ottobre 2009**

L'obiettivo di un sistema basato su microprocessore o DSP è quello di arrivare alla conoscenza di una struttura capace di miscelare segnali appartenenti al campo analogico con segnali appartenenti al campo digitale.




La struttura capace di gestire dati e compiere operazioni matematiche deve essere in grado di gestire e comunicare con il mondo esterno.

Sistemi basati su microprocessore, microcontrollori e DSP ricalcano la stessa filosofia di fondo.

le distinzioni riguardano essenzialmente le specializzazioni rispetto a certe funzioni assolte.

Le funzioni gestite sono direttamente legate al costo del dispositivo ed alla occupazione di area necessaria alla realizzazione del dispositivo.

Queste distinzioni si fanno sempre meno accentuate, tanto che stanno cominciando ad apparire sul mercato i così detti DSC, ovvero Digital control Processor.

# L'organizzazione della memoria 1

I sistemi basato su microcontrollori e microprocessori si distinguono per una caratteristica fondamentale: l'organizzazione della memoria.

I sistemi basati su microprocessori, come ad esempio i Personal Computer, dispongono di una singola memoria dove sono posti sia i dati che i programmi. La memoria risulta ovviamente strutturata in modo che si possa distinguere tra le 2 strutture.

Tuttavia lo scambio dei dati, delle istruzioni e degli indirizzi tra l'unità di elaborazione e la memoria avvengono attraverso dei canali comuni. Questa soluzione sicuramente ottimizza i costi, ma, per contro, non risulta ottimale nell'ottimizzazione della velocità di trasmissione.

Von Neumann

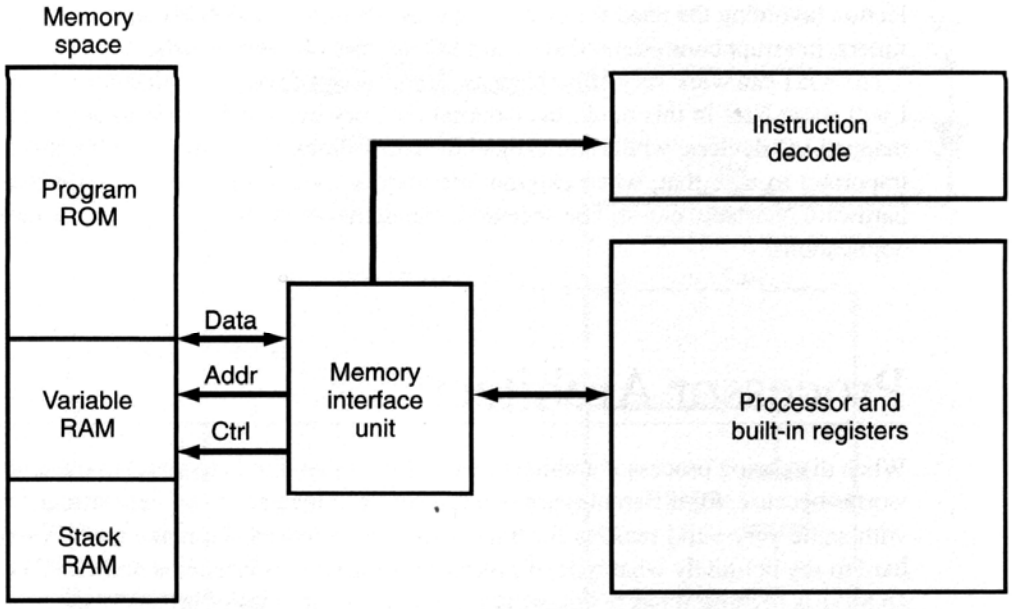



**FIGURE 1-4** Princeton architecture block diagram.

Un esempio: lettura di un dato dalla memoria.

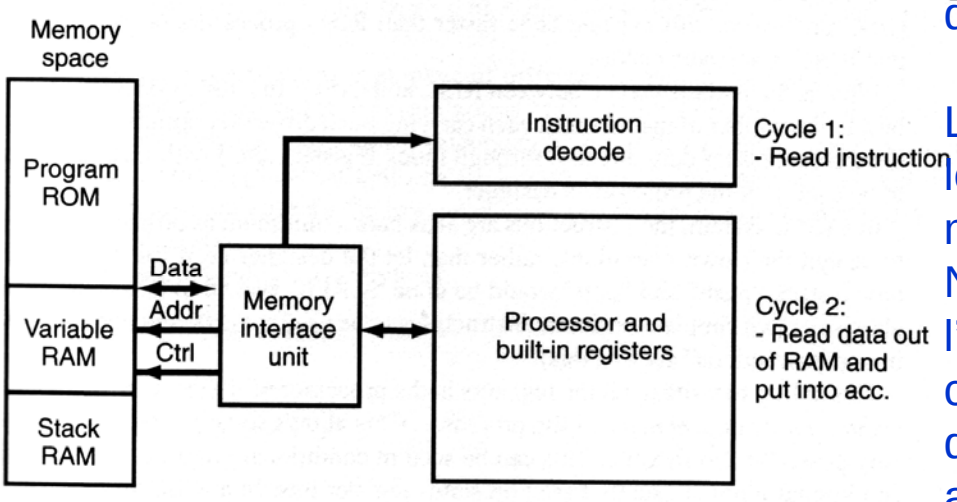



**FIGURE 1-6** `mov Acc, Reg` in the Princeton architecture.

La prima fase riguarda la lettura della istruzione dalla memoria.
Nella seconda fase l'esecuzione dell'istruzione consiste nella lettura del dato dalla memoria stessa attraverso lo stesso bus.

# L'organizzazione della memoria 2

I microcontrollori ed i DSP hanno, di regola, la memoria organizzata secondo lo schema così detto Harvard: la memoria programmi e la memoria dati sono separate.
Questo comporta la presenza di un sistema a doppio bus: un bus indirizzi per la memoria programmi ed il relativo bus istruzioni, ed una bus indirizzi per la memoria dati con il relativo bus dati.
Questa soluzione è molto comoda visto che i sistemi a microcontrollori operano con un solo programma che in questo modo viene "stoccato" in una apposita memoria non-volatile.

Va menzionato che sebbene tutti dispongano di separate memorie dati ed istruzioni, non tutti i microcontrollori dispongono anche del sistema a doppio bus. L'efficienza viene meno, ma la separazione tra dati ed istruzioni risulta comunque utile.

Harvard

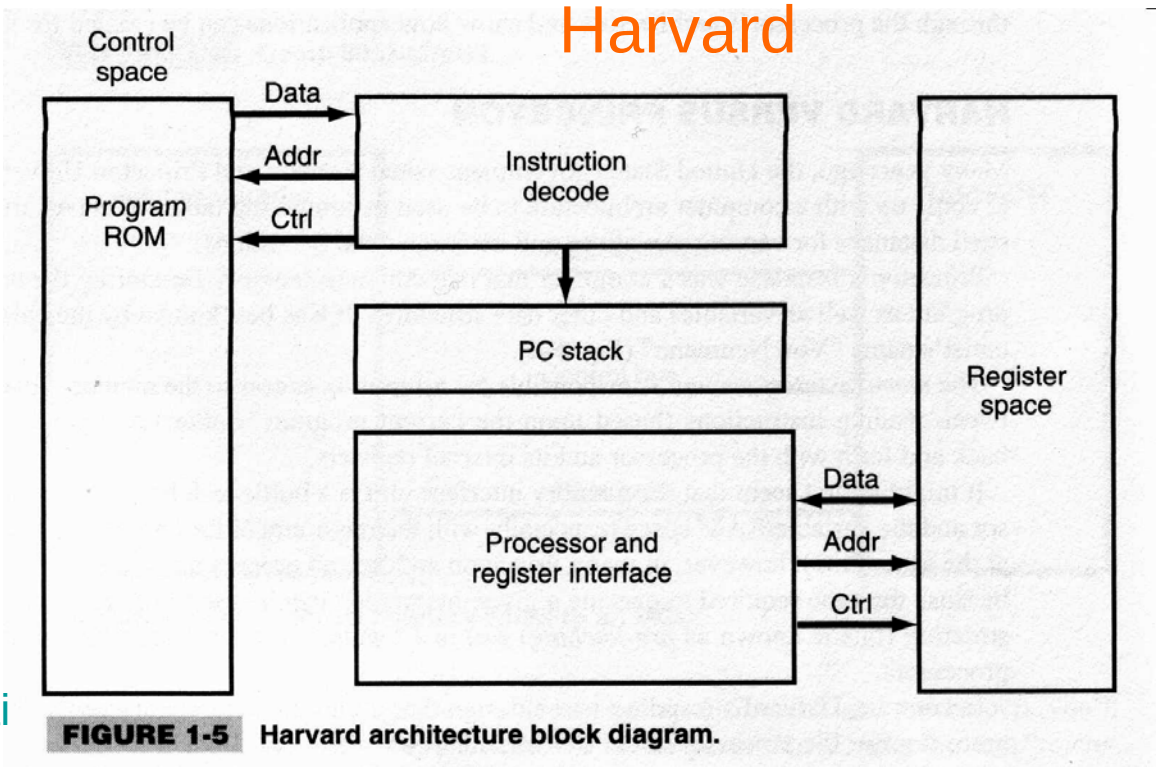



**FIGURE 1-5** Harvard architecture block diagram.

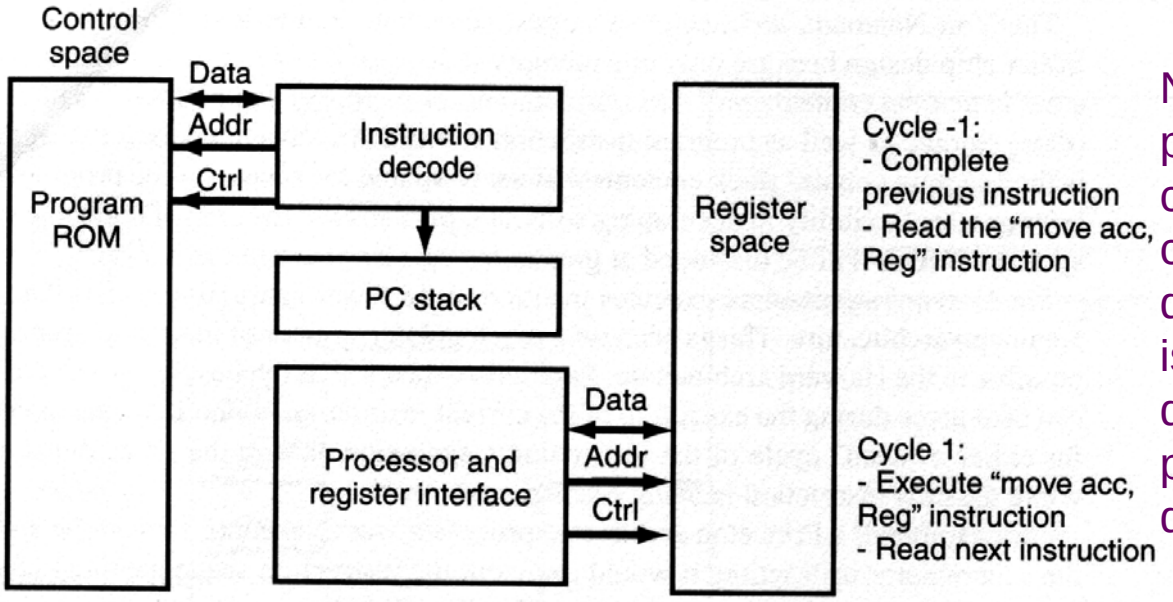



**FIGURE 1-7** `mov Acc, Reg` in the Harvard architecture.

Nell'esempio si può osservare come in un singolo ciclo si possa disporre sia della istruzione che del dato grazie alla presenza del doppio bus.

# Struttura del microcontrollore 1

Lo schema più semplificato di una struttura a microprocessore è suddivisibile in 3 blocchi:

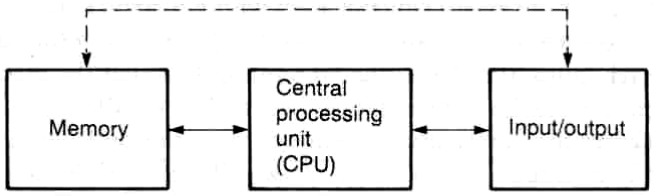



**Figure 12.1** The main sections of a computer.

La memoria è la sede dei programmi e dei dati. I sistemi di comunicazione di ingresso / uscita consentono la interfaccia con il mondo esterno. L'unità di controllo gestisce il funzionamento del sistema mediante la scansione delle operazioni da svolgere e tutte le procedure matematiche del caso.

Seguendo l'approccio top down il primo aspetto da sviscerare riguarda la modalità di comunicazione tra le 3 parti principali individuate.

I blocchi comunicano tra loro mediante un sistema di bus, ovvero un insieme di linee elettriche parallele nelle quali scorrono informazioni omogenee.

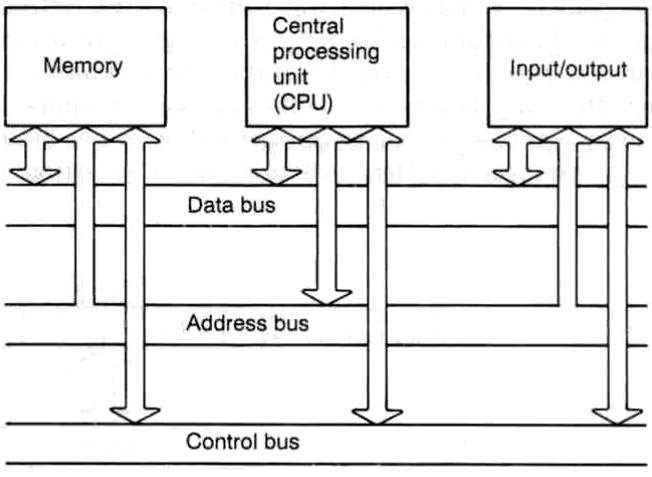



**Figure 12.2** A typical microcomputer bus system.

Serve almeno un bus indirizzi (2 nel caso i dati siano separati dal programma) che serve a decidere quale dato deve essere disponibile al momento necessario.

Serve almeno un bus (2 nel caso si tengano separati i dati dalle istruzioni) dati dove i dati selezionati devono viaggiare.

Un bus di controllo, in genere sottointeso negli schemi, attraverso cui vengono selezionate le unità che devono comunicare / operare quando selezionate. Il bus di controllo è composto da linee non tutte in comune, che conettono singoli elementi.

# Struttura del microcontrollore 2

La connessione dei vari elementi ai bus avviene mediante l'abilitazione delle uscite dallo stato di alta impedenza. La tipica connessione avviene attraverso un registro che viene posto nelle condizioni di scrivere il suo contenuto in qualche altro registro passando attraverso il bus dati.

Il registro è la più piccola unità di memoria che soddisfa il parallelismo dell'unità: 8 bit, 16 bit, 32 bit, ecc. Un micro ad 8 bit è predisposto per elaborare dati di 8 bit, pur potendo indirizzare a 16 bit ed anche a 24 bit.

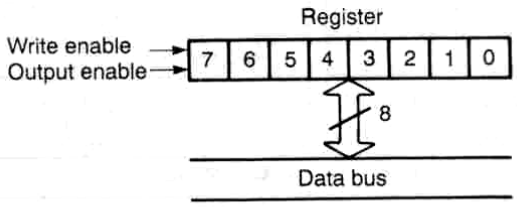



**Figure 12.3** A simple 8-bit register.

La memoria dati o programmi è composta da molti registri impilati. Ogni registro è indirizzabile. I registri contenuti nelle memorie vengono detti celle di memoria.

La definizione di registro la si usa per indicare registri speciali o i contenuti di piccole memorie (tipicamente dati).

In genere i singoli bit di una cella di memoria o registro non sono elaborabili singolarmente eccetto che per alcuni registri speciali che contengono informazioni del sistema.

Esistono 2 tipologie di memoria diverse usate nei micro:

RAM: I dati sono contenuti in una memoria volatile di accesso veloce sia in lettura che scrittura.

ROM: memoria programmi scrivibile una sola volta e leggibile sempre.

FLASH: la memoria programmi più in voga. Si scrive lentamente ma l'operazione di scrittura può essere fatta un numero elevato di volte. La lettura è veloce.

# Struttura del microcontrollore 3

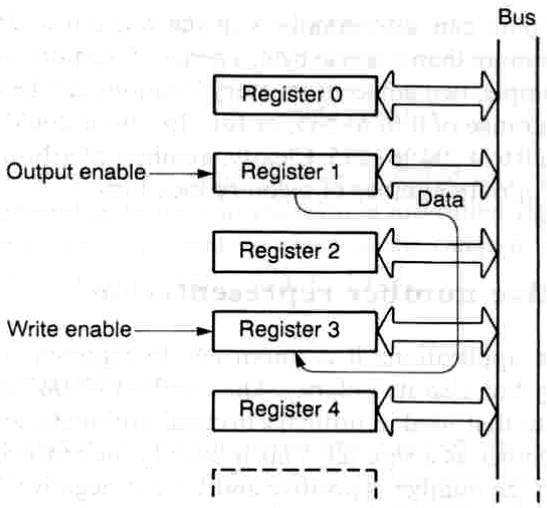



**Figure 12.4** Communication between registers.

Un esempio di trasferimento diretto di dati tra 2 registri.

Il registro 1 è indirizzato ed abilitato a scrivere sul bus dati.

Il registro 3 è indirizzato ed abilitato a ricevere il dato dal bus dati.

In genere il passaggio diretto di dati tra registri è ammesso. Non è ammesso il passaggio diretto di dati tra 2 celle di una memoria. Questo perché 2 celle non possono essere selezionate contemporaneamente.

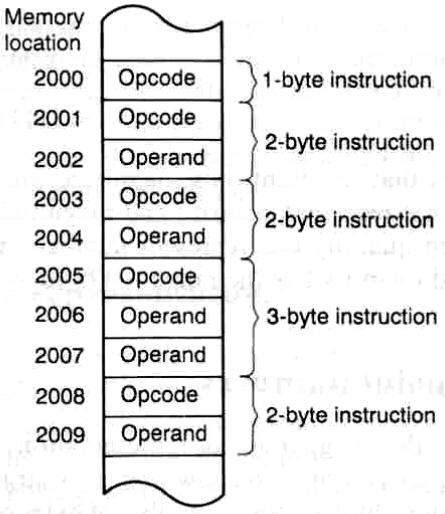



**Figure 12.5** Storage of a program.

La memoria programmi è una pila di celle di memoria ognuna contenete un codice operativo. Durante il funzionamento del sistema sequenzialmente i contenuti delle celle vengono letti ed interpretati come comandi.

Ovviamente la lettura della memoria non è necessariamente consecutiva. Sulla base dei risultati e degli eventi che si verificano il sistema può decidere di eseguire certi salti, predisposti, a determinate zone di programma.

# Struttura del microcontrollore 4

Qui vediamo un primo esempio, semplificato, di micro completo: l'MC6800 di Motorola.

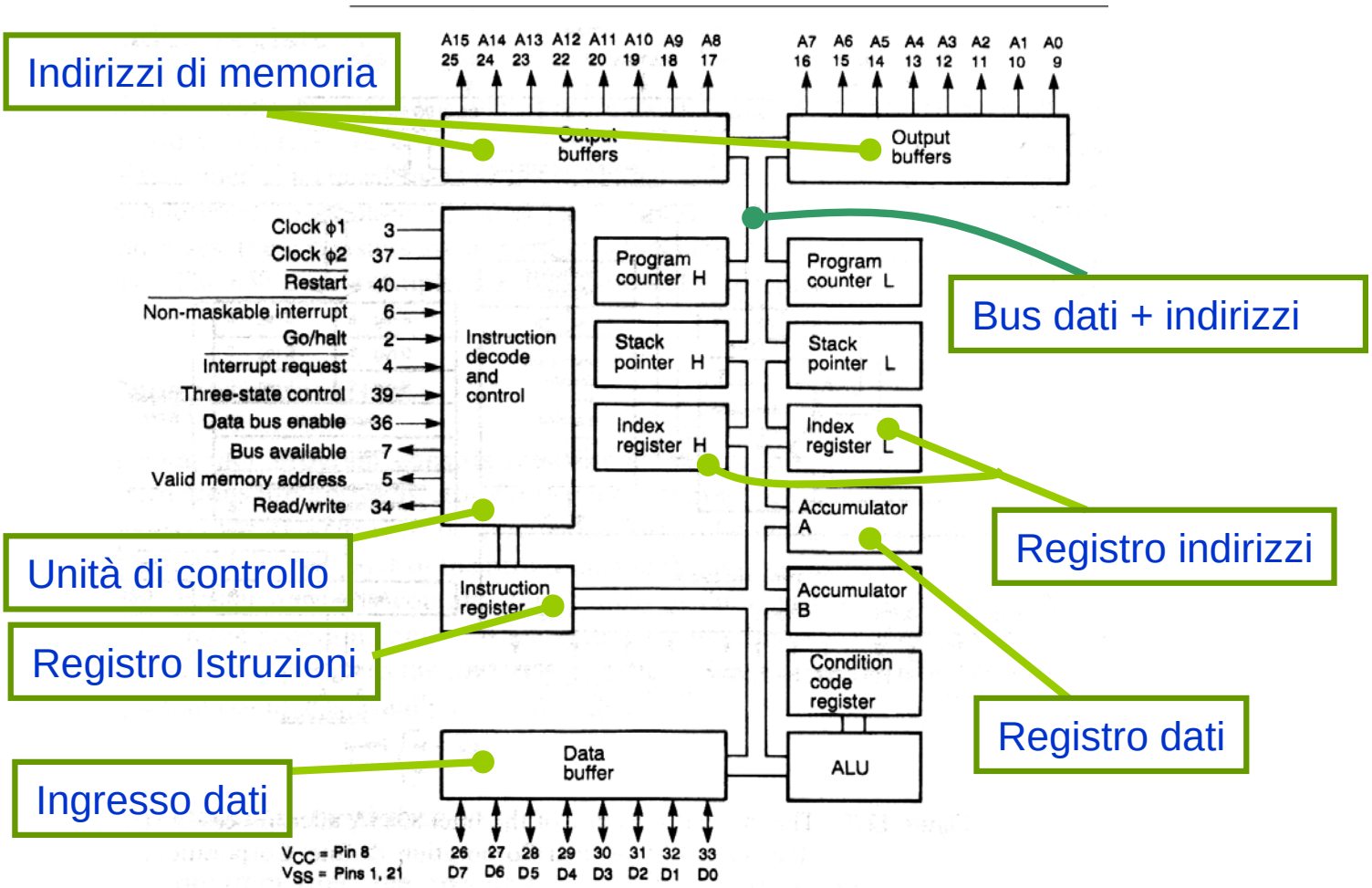



**Figure 12.6** The internal structure of the Motorola MC6800 microprocessor.

L'MC6800 è un tipico microprocessore. La sua comunicazione con il mondo esterno avviene attraverso i dati e le istruzioni che stanno nella memoria esterna.
All'interno la memoria contiene solo alcuni registri.
I registri indirizzi si distinguono sempre perché sono divisi in byte (8 bit) ed hanno una parte bassa, L, ed una parte alta, H.
I registri dati sono a singolo byte (l'MC6800 è a 8 bit).

In questo schema semplificato non viene indicata marcatamente la separazione tra bus dati ed indirizzi. Spesso questa semplificazione viene assunta perché ritenuta inessenziale all'utilizzatore.

# Struttura del microcontrollore 5

Anche qui abbiamo un microprocessore che non possiede memoria interna. La sua schematizzazione è più completa.

Il bus indirizzi viene distinto in questo caso dal bus dati a livello di buffer esterno.

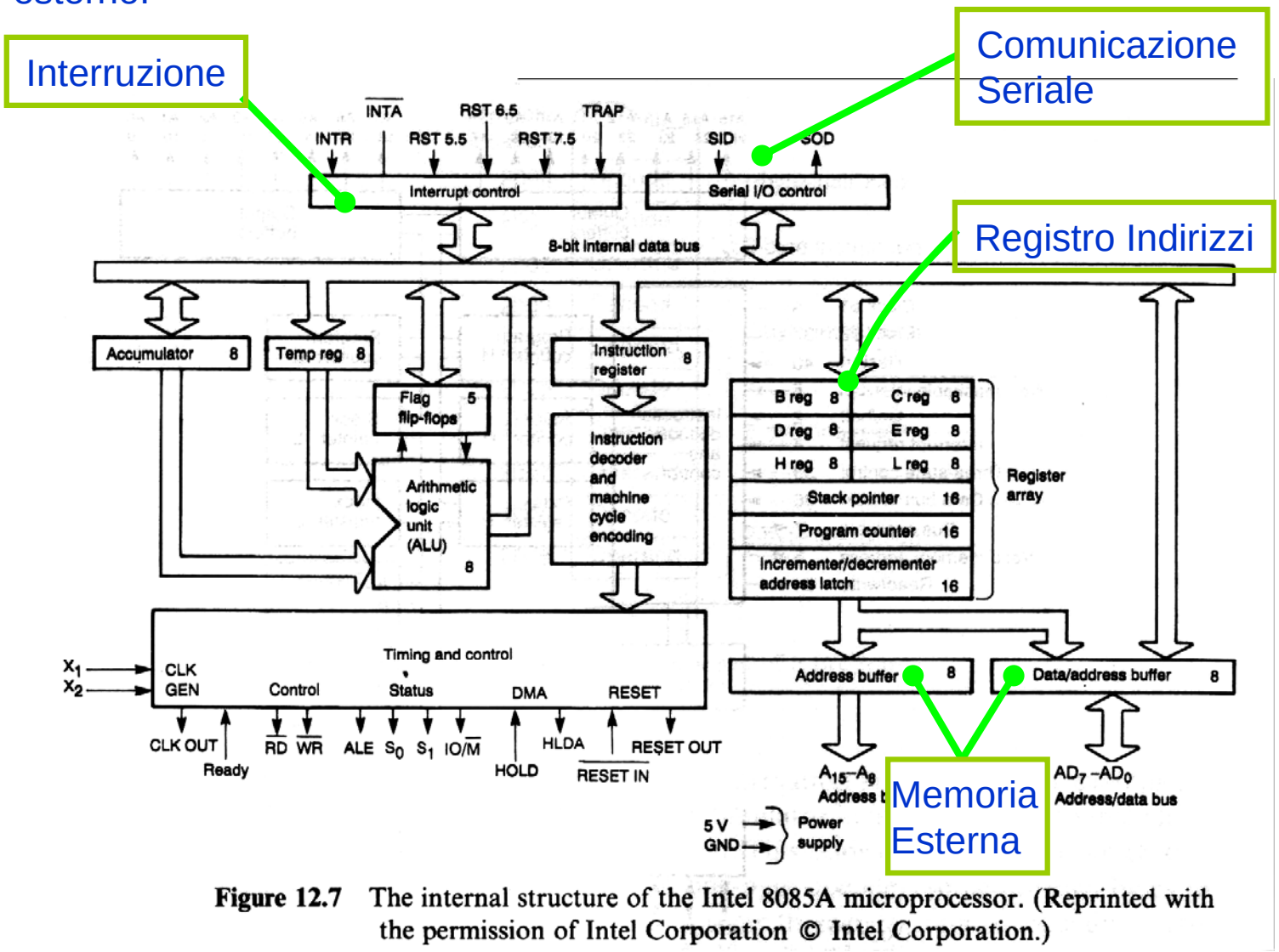



**Figure 12.7** The internal structure of the Intel 8085A microprocessor. (Reprinted with the permission of Intel Corporation © Intel Corporation.)

Si distingue già una differenza rispetto all'MC6800. La memoria viene letta con solo 2 byte. Serve un ausilio esterno per indirizzare 2 byte. Comunque la lettura deve essere svolta in 2 passi.

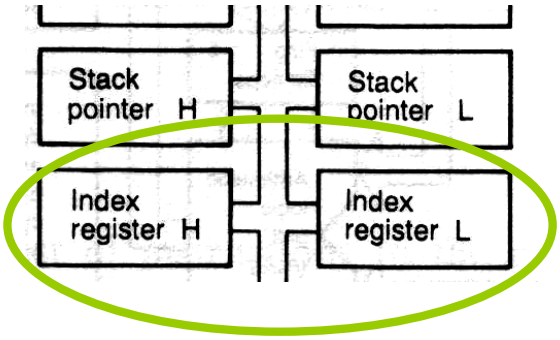
Si distingue la parte di elaborazione matematica, ALU, con il registro accumulatore, A, ed il registro temporaneo, B.

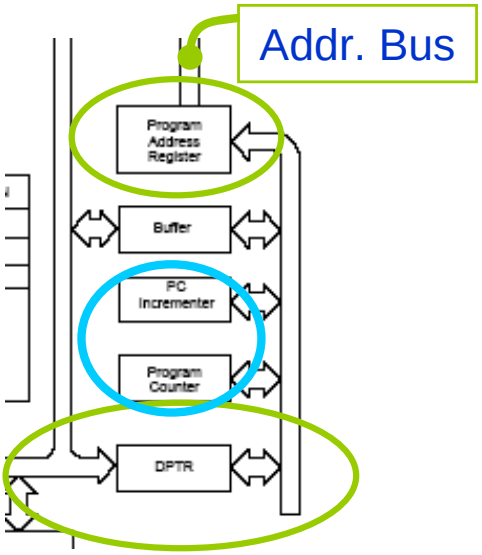
L'unità di controllo viene messa in evidenza in connessione con il registro di decodifica delle istruzioni.

# Struttura del microcontrollore 6

Vi sono degli elementi fondamentali che fanno parte del cuore di ogni microcontrollore o microprocessore.

**Index registers**: sono registri generici dove si possono contenere indirizzi che possono essere utilizzati. Un registro molto importante in molti micro con struttura Harvard o Harvard modificata è il **DPTR o Data Pointer**. Questo registro contiene l'indirizzo dei dati contenuti nella memoria dati esterna.




**Program Counter, PC**: questo registro contiene l'indirizzo della successiva istruzione da eseguire, da prelevare dalla memoria programmi, se si sta procedendo in modo sequenziale. Altrimenti conterrà l'indirizzo della cella a cui si dovrà saltare.

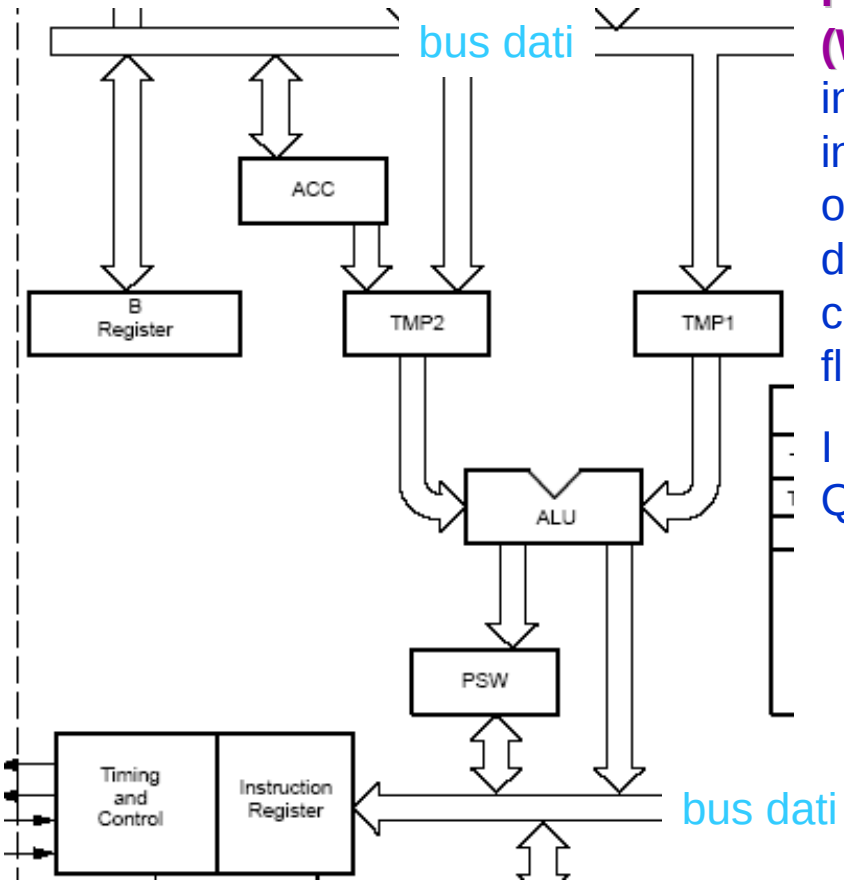
**Instruction Register**: contiene l'istruzione da eseguire, letta dalla memoria puntata dal PC. Questo registro è connesso direttamente con il decodificatore di istruzione, che converte il codice dell'istruzione in segnali da inoltrare alle varie parti del micro, e con l'unità di controllo del micro con il mondo esterno.

# Struttura del microcontrollore 7

**Arithmetic and Logic Unit**, ALU: qui si svolgono tutte le operazioni matematiche. Anche le operazioni logiche vengono affrontate qui dentro. Possono essere anche eseguiti, scorrimenti, shift dei dati, etc.

**Accumulatore, A**: è il registro fondamentale. Contiene un operando di tutte le operazioni matematiche. Spesso anche il risultato dell'operazione matematica svolta. Molti movimenti di dati vengono svolti con l'Accumulatore come registro di passaggio (appoggio).

Esiste un secondo registro, **B**, che svolge la funzione di registro ausiliario nelle operazioni di moltiplicazione e divisione.

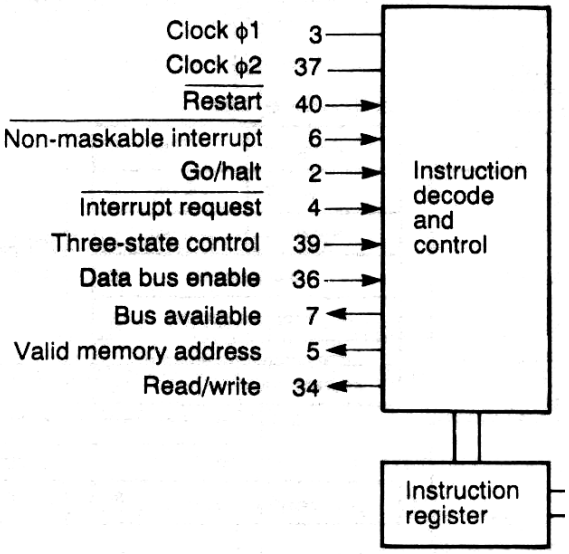



**Processor Status Register (Word), PSR (PSW):** i bit contenuti in questo registro forniscono le indicazioni del risultato delle operazioni matematiche svolte o dei confronti effettuati. I bit contenuti qui dentro vengono detti flag.

I flag cambiano da micro a micro. Quelli comuni sono 4:

- **Z**: impostato a 1 se il risultato nell'ALU è zero;
- **N**: impostato a 1 se il risultato dell'ALU è negativo;
- **C**: riporto di una somma;
- **V**: overflow risultato troppo grande;

# Struttura del microcontrollore 8




## Unità di controllo e decodifica di istruzione

Questo è il cuore del micro. A questa unità è connesso il clock che ne scandisce la sequenza di operazioni.

Possiamo pensare il clock diviso in 2 cicli composti ciascuno da più colpi di clock.

Il primo ciclo è la ricerca dell'istruzione o instruction fetch. Il secondo ciclo è l'esecuzione dell'istruzione.

La fase di fetch può seguire 2 modalità: la prima riguarda il caricamento dell'istruzione dalla memoria interna al micro, la seconda il caricamento dell'istruzione dalla memoria esterna.

In entrambi i casi si tratta di fornire un comando al PC di scrittura dell'indirizzo, e di abilitazione di lettura dalla memoria, esterna o interna, dell'istruzione. Il numero di colpi di clock necessari potrà essere differente se la memoria indirizzata è interna o esterna. Infatti nel secondo caso occorre passare attraverso i buffer di uscita/ingresso.

La fase di esecuzione dell'istruzione può impiegare un numero di passi che può essere molto variabile in funzione della complessità delle operazioni da svolgere: devono essere generati i segnali di controllo per le unità che devono essere interessate dalla istruzione.

Potrebbe essere necessario dovere caricare dei dati nella ALU per compiere operazioni o confronti. Ad esempio le operazioni di moltiplicazione e divisione richiedono il massimo sforzo temporale di diversi colpi di clock.

La velocità di operazione del micro non è misurata dal clock ma dal numero di colpi di clock generalmente usati nell'esecuzione di ogni istruzione (**Ciclo Macchina**). Ci sono micro che impiegano 2 copi di clock, 4 colpi di clock ed anche 12 colpi di clock.

# Che cosa è l'unità di controllo

Un microcontrollore non è nient'altro che una macchina a stati. L'unità di controllo è il cuore di questa macchina a stati sequenziale.

L'esecuzione di un'istruzione implica semplicemente che deve essere eseguita una successione di azioni che necessitano il passaggio attraverso differenti stati.

Il passaggio tra uno stato e l'altro è cadenzato dal clock.

Istruzioni complesse utilizzano molte azioni, quindi il passaggio attraverso molti stati, da qui l'alto numero di colpi di clock.

Operazioni semplici necessitano pochi colpi di clock.

Il successo dei microcontrollori risiede nel fatto che le macchine a stati che li compongono sono state progettate in modo da seguire standard che le rendono adattabili a molte applicazioni e adottati da molti progettisti aventi diverse necessità.

Microcontrollori di diverse case costruttrici hanno caratteristiche differenti. Però il cuore di ogni macchina è molto simile, con i registri fondamentali sempre presenti e ricoprenti lo stesso ruolo.

# Struttura del microcontrollore 9

Un esempio di sequenza di cicli macchina per il classico micro Intel 8051: in origine questo micro aveva bisogno di 12 colpi di clock (6 per il primo byte e 6 per il secondo byte) per eseguire un'istruzione. Perciò con un clock di 12 MHz il micro poteva eseguire mediamente 1 istruzione al μsec.

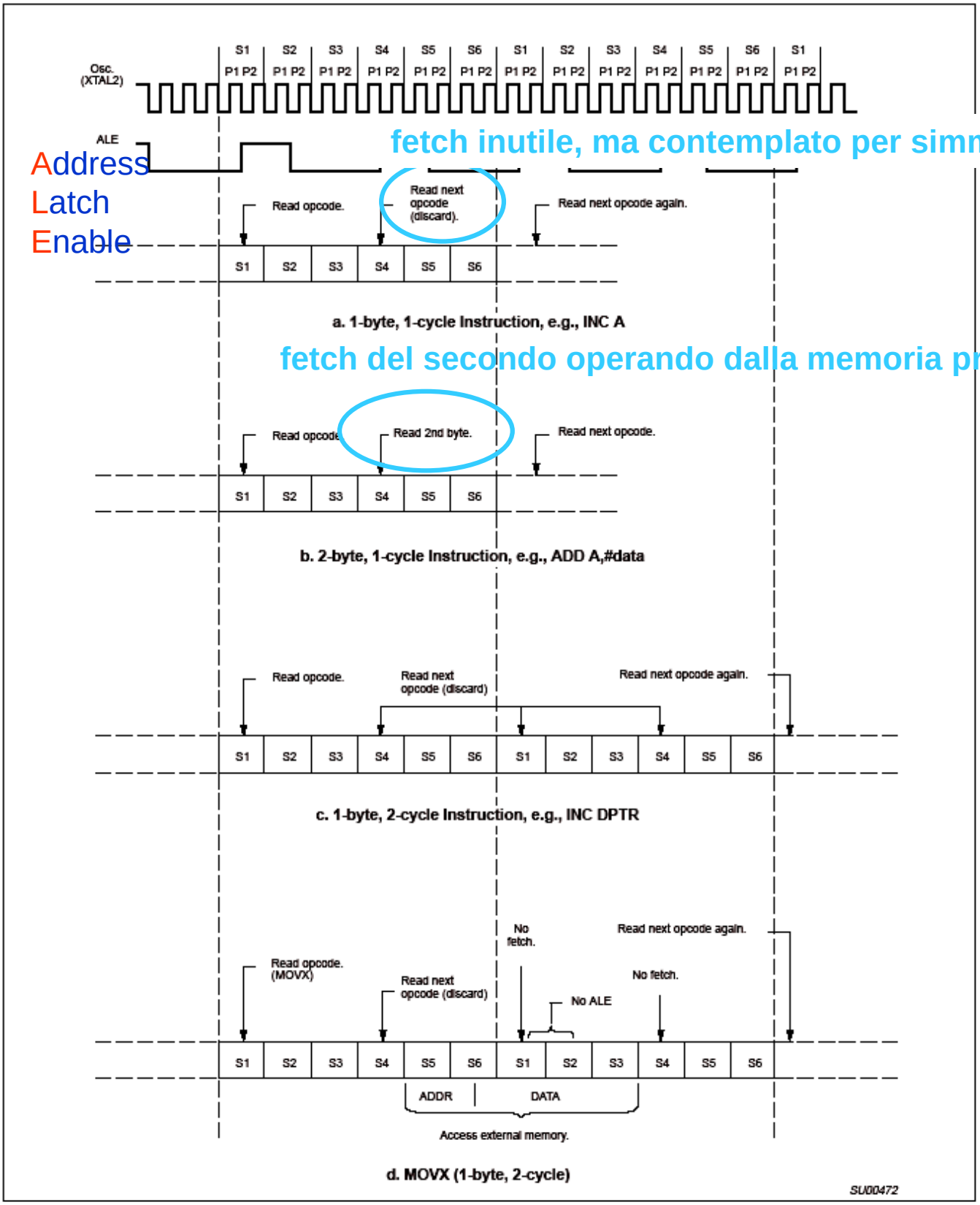



Figure 15. State Sequence in 80C51 Family Devices

# Struttura del microcontrollore: la memoria 1

La lettura dalla memoria esterna merita qualche dettaglio.

Prima di tutto la memoria necessita di diversi comandi per essere messa nelle condizioni di dialogare.

La memoria va abilitata, ovvero le uscite vanno tolte dalle condizioni di alta impedenza e devono diventare di ingresso se viene inviato il segnale di WE (write enable), o di uscita se viene inviato il segnale OE (output enable da parte della memoria).

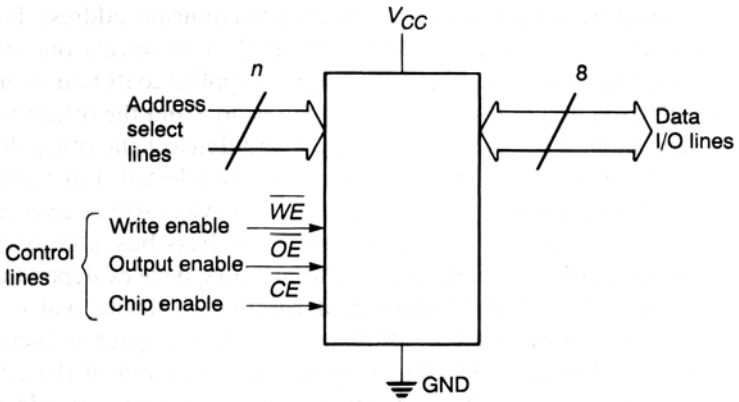



**Figure 12.13** A typical memory device.

In caso più memorie siano usate la linea CE consente la selezione della memoria stessa.

La situazione più complicata la abbiamo quando le linee dedicate alla memoria sono 16, ma servono 16 bit per l'indirizzo e 8 per il dato. Occorre avere l'ausilio di una memoria addizionale, o latch, che memorizzi una parte dell'indirizzo, di solito quello basso.

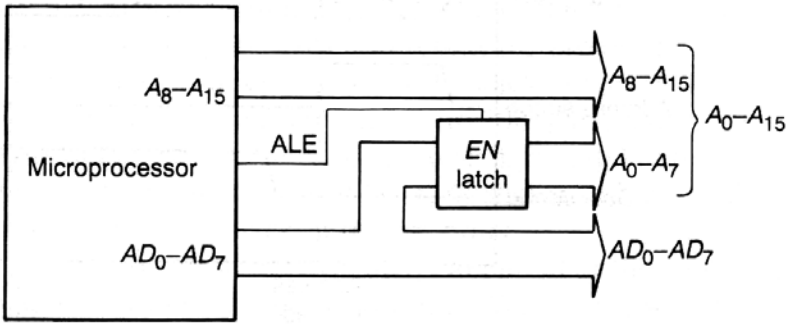



**Figure 12.12** The use of an address latch.

La lettura o scrittura nella memoria avviene in 2 fasi. La prima serve per memorizzare la parte inferiore dell'indirizzo nel latch. La seconda per la lettura / scrittura del dato.

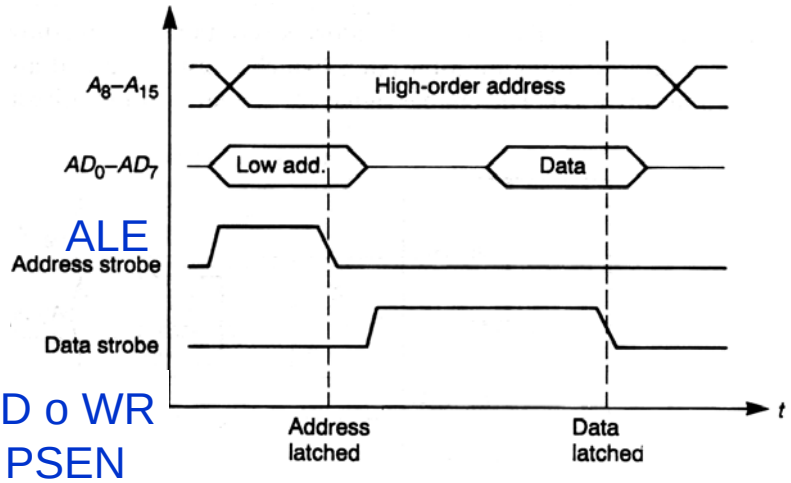



**Figure 12.10** Bus multiplexing.

# Struttura del microcontrollore: la memoria 2

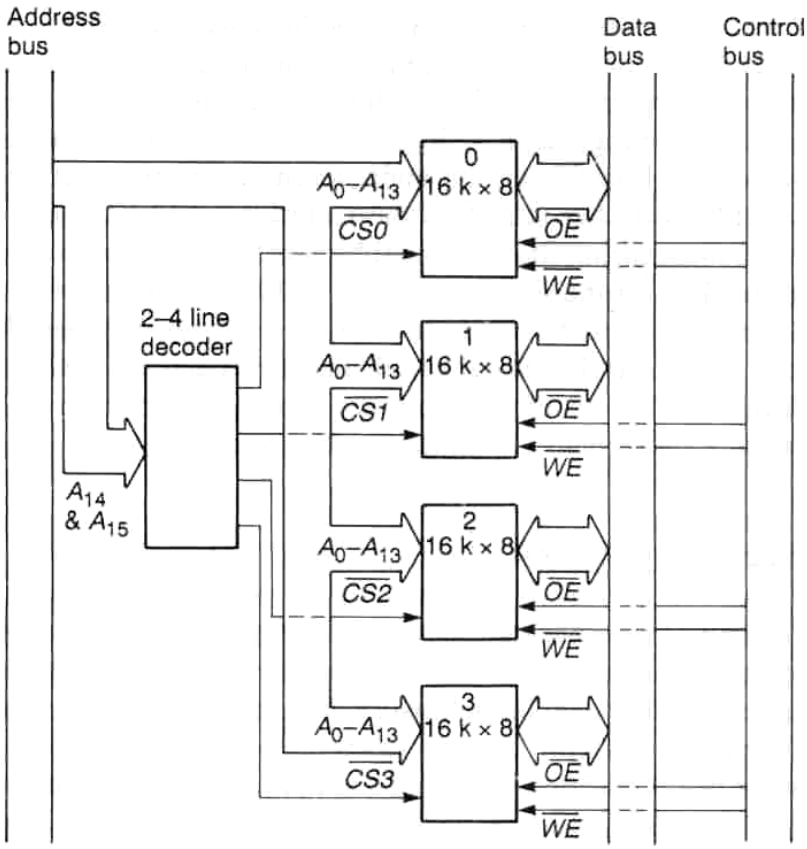



**Figure 12.14** A typical memory arrangement.

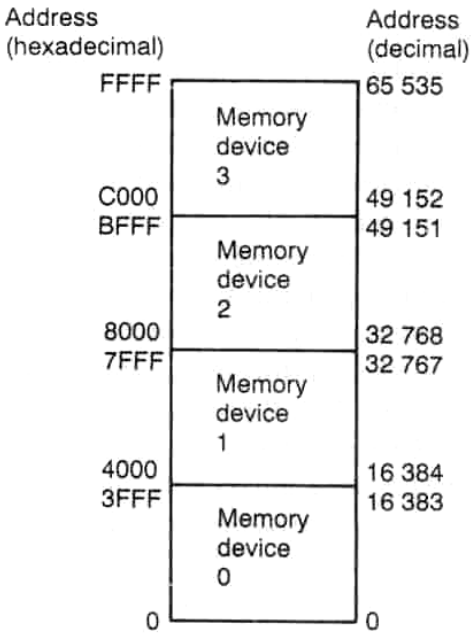
Si supponga di volere indirizzare 64KB di memoria. L'indirizzo è a 16 bit.

Si abbiano 4 banchi da 16KB l'uno, quindi indirizzabili con 14 bit.

I 2 bit più significativi dei 16 di indirizzo vengono allora usati per pilotare un decoder che seleziona 1 di 4 possibili linee in uscita. L'uscita seleziona sarà l'ingresso CS, chip select, della memoria necessaria.

Es.: l'indirizzo 17000d sarà localizzato nella memoria 1 nella cella 611=17000-16383 ($16383=2^{14}-1$)

17000d=01 00001001101000b, 01b del decoder seleziona la prima linea




La memoria viene rappresentata come una pila di 4 banchi.

Questo tipo di organizzazione viene detto Memory map.

# Struttura del microcontrollore: la memoria 3

Per esempio si può osservare la lettura dell'istruzione dalla memoria programmi quando posta esterna al micro.

Il PCH sono gli 8 bit più significativi del PC, mentre PCL sono gli 8 bit meno significativi.

Nella parte superiore della figura viene solo letta l'istruzione dalla memoria esterna programmi. Siccome è una memoria di sola lettura, la linea RD non viene cambiata.

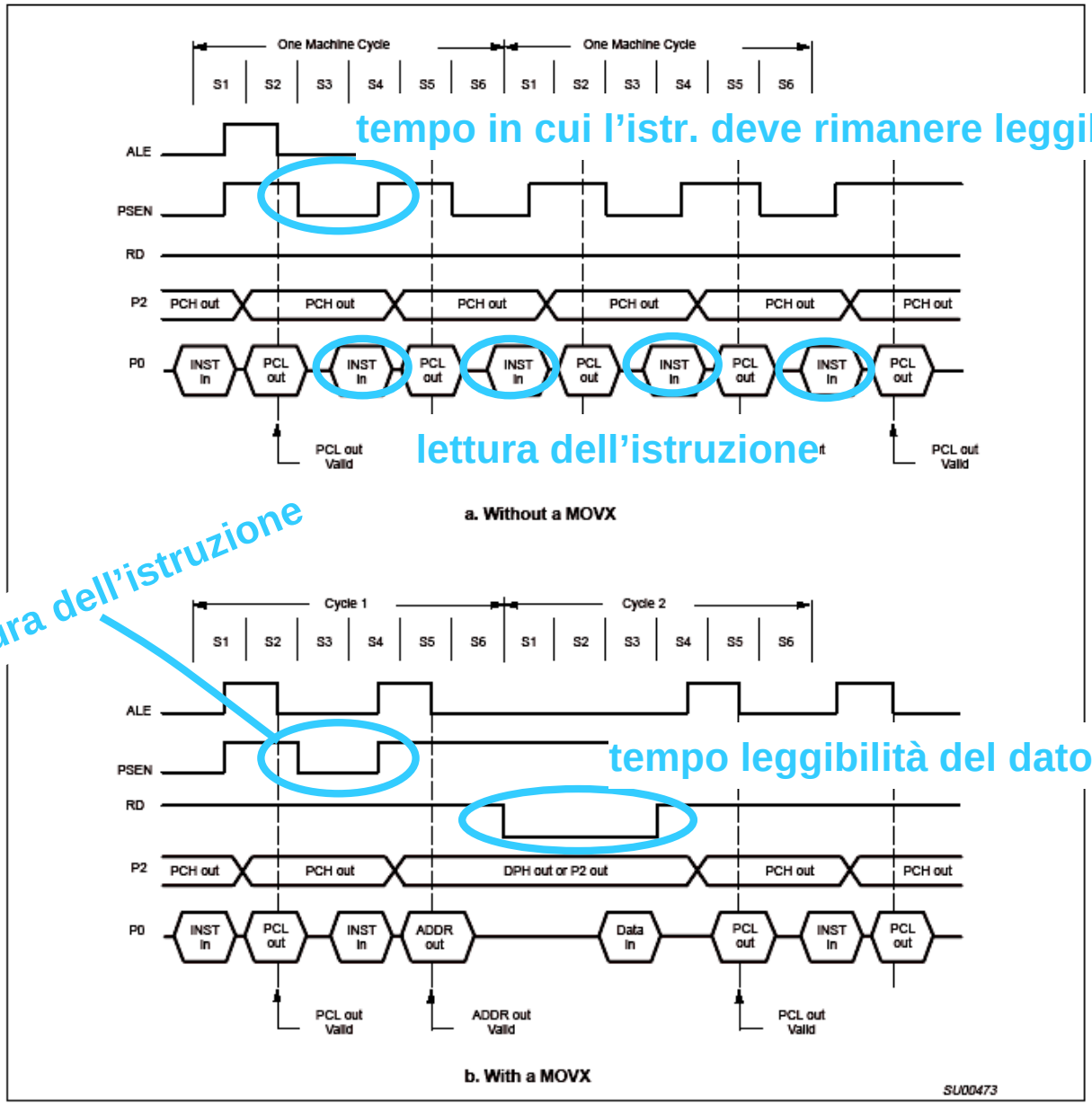



Figure 16. Bus Cycles in 80C51 Family Devices Executing from External Program Memory

Nella parte inferiore della figura vengono lette l'istruzione e la memoria dati, entrambe esterne, consecutivamente. In questo caso quando si legge il dato la linea di RD viene abbassata. Questo perché la memoria dati esterna è anche di scrittura.

# Struttura del microcontrollore: la memoria 4

Vediamo l'aspetto HW di come la memoria esterna per dati e programmi potrebbe essere connessa al micro.

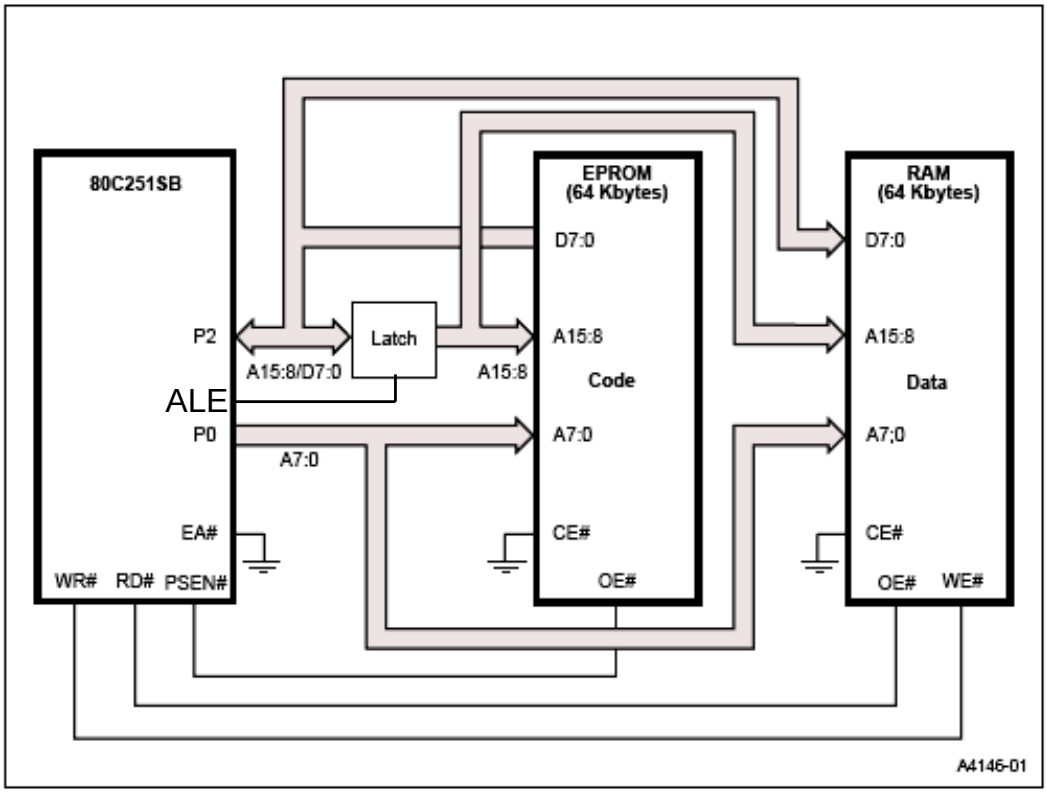



Figure 12-18. 80C251SB in Page Mode with External EPROM and RAM

La memoria programmi qui è una EEPROM. Siccome gli OE e WE sono linee separate, il pin CS, o CE, delle 2 memorie in questa situazione è ridondante.

Il pin EA del micro viene tenuto basso se tutta la memoria programmi si desidera che risieda esternamente.

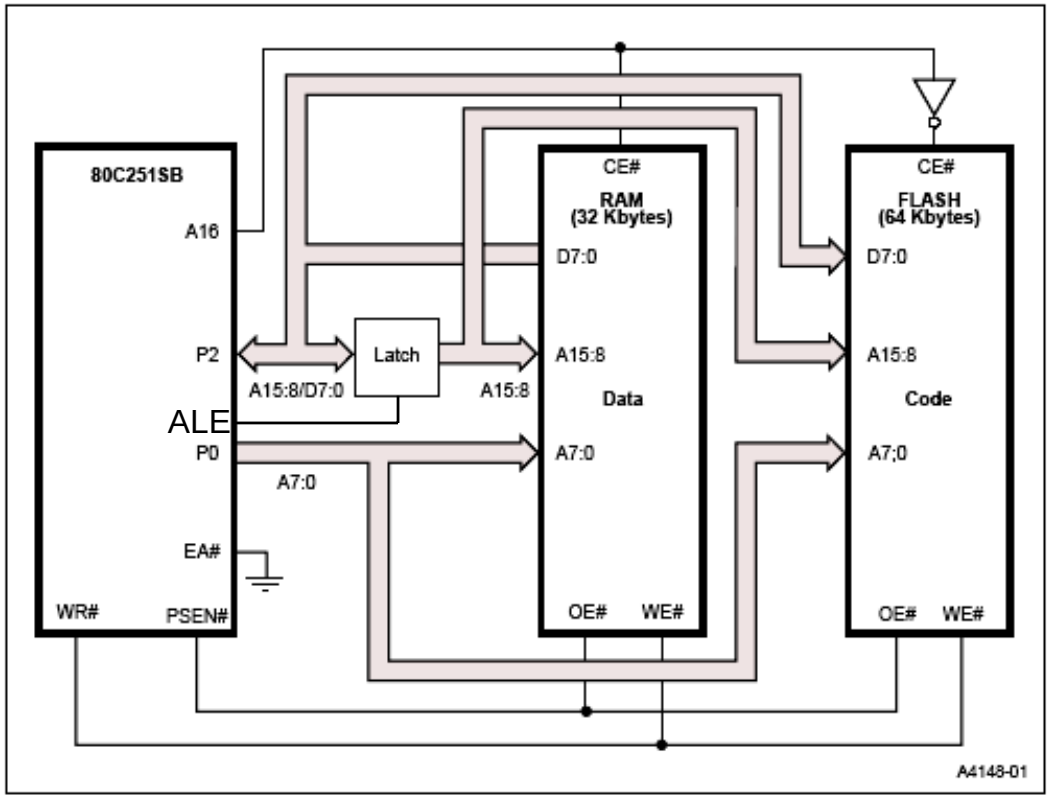
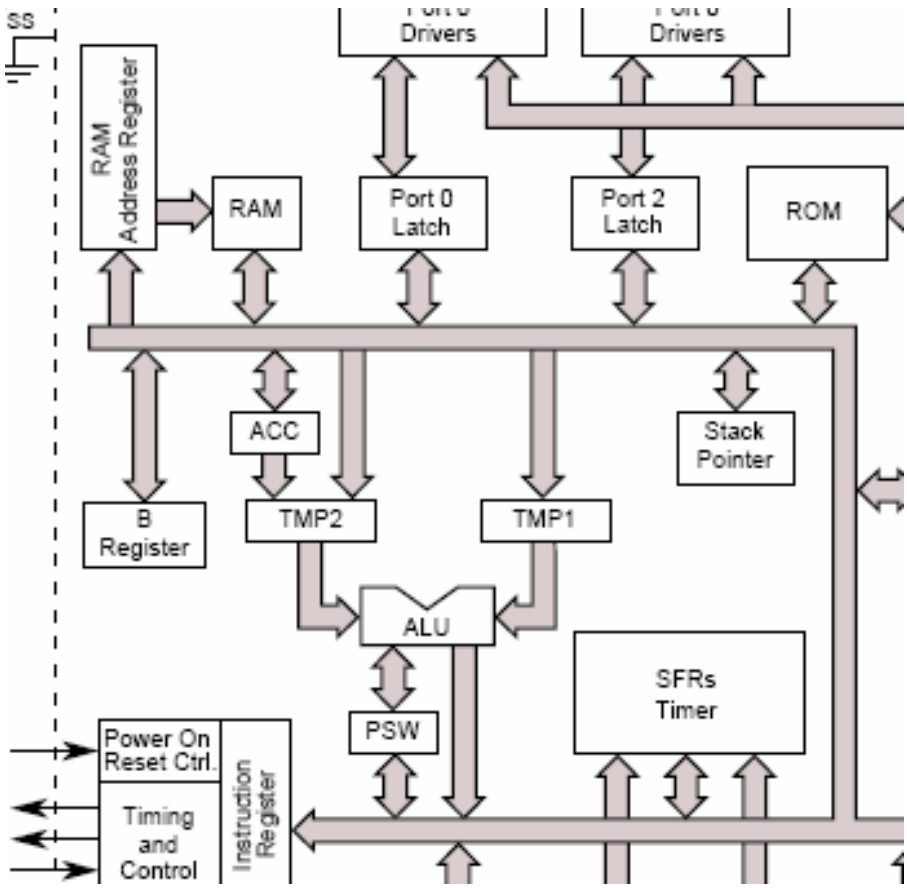



Figure 12-19. 80C251SB in Page Mode with External Flash and RAM

Se la memoria esterna è una FLASH, volendo aggiornare il programma "in-circuit" occorre utilizzare un bit addizionale per i 2 CS, o CE. Questo bit viene usato solo quando si deve aggiornare il programma.

# Struttura del microcontrollore: la memoria 5

Un comportamento simile avviene nell'operazione di movimentazione dei dati o istruzioni all'interno del micro stesso.



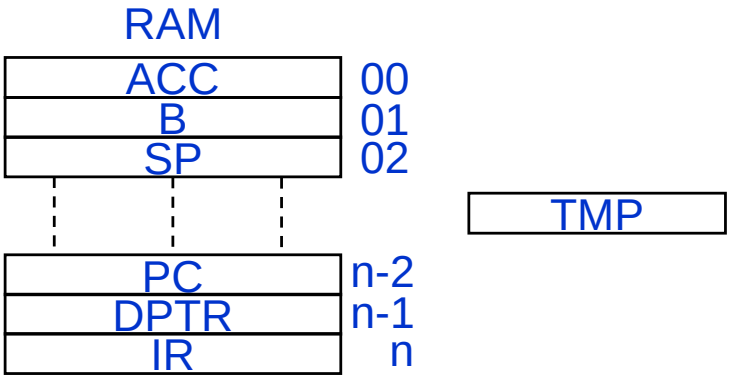

I registri ACC, B, ecc. sono tutti impilati nella RAM. Solo che i loro indirizzi sono assegnati ed associati alle funzioni a cui sono adibiti.

Il trasferimento tra RAM e RAM, o tra registro e registro, o tra RAM e registro necessita di 2 passaggi perché la RAM ed i registri devono essere nella stessa modalità allo stesso tempo.

Ovvero dobbiamo disporre di almeno un latch, per esempio TMP2, che non faccia parte della RAM.

L'implementazione della trasmissione: RAM(i)→ACC potrebbe avvenire così:

1. RAM(i) posta in lettura;
2. TMP posto in scrittura;
3. RAM(i) disabilitata;
4. ACC in scrittura;
5. TMP posto in lettura.




Si deve osservare che il processo coinvolge il passaggio del dato attraverso la ALU. In genere la movimentazione dei dati nei micro coinvolge sempre il passaggio attraverso la ALU e ACC.

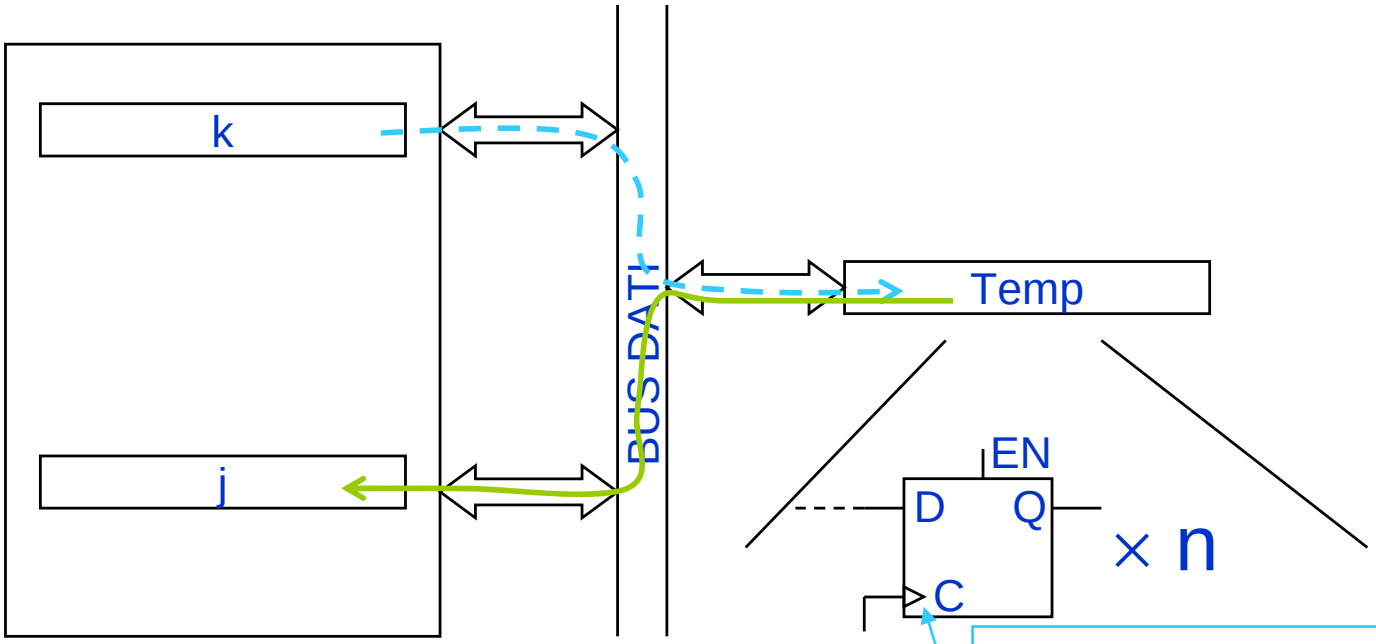
# Struttura del microcontrollore: la memoria 6

L'aggiunta del registro TMP, e la possibilità di porre l'uscita ad alta impedenza, ovvero sconnessione dal circuito consente la realizzazione del trasferimento di dati tra celle di memoria.

ES.: Trasferimento di dato tra la cella all'indirizzo K e la cella all'indirizzo j.

- - - - → Il dato dalla cella k viene scritto nel registro temporaneo abilitato in scrittura. Il registro temporaneo opera sul fronte di discesa del clock. Il trasferimento del dato tra Temp e memoria (e viceversa) avviene sul fronte del clock, non sullo stato dello stesso. Questa condizione assicura la stabilità del dato.

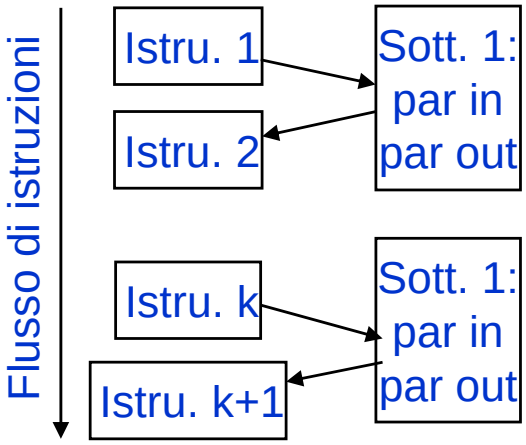
——→ Il registro Temp viene abilitato alla lettura dal bus dati. Il registro j viene abilitato alla scrittura: si scrive il dato nella memoria alla cella desiderata.




Il triangolino significa che questo registro memorizza sul fronte si salita del clock, non sul livello.

# Il servizio ai sottoprogrammi: lo stack 1

I sottoprogrammi sono quegli insiemi di istruzioni più o meno complessi che svolgono funzioni comuni a molte parti del programma: si chiama la routine passando i parametri necessari. In risposta si ricevono le elaborazioni svolte:




Nel linguaggio ad alto livello l'unica cosa di cui occorre tenere conto sono i parametri da passare al sottoprogramma e quelli che si riceveranno. I parametri usati all'interno del sottoprogramma non devono essere visibili all'esterno e viceversa.

Quando si considera la programmazione assembler le stesse proprietà devono ancora valere. Solo che occorre costruirsele: l'hardware è comune ad entrambe le sequenze di istruzioni.

Per risolvere questa sorta di "collo di bottiglia" si ricorre all'uso dello Stack.

Lo Stack è una pila di registri che vengono indirizzati con la modalità definita last-in-first-out.

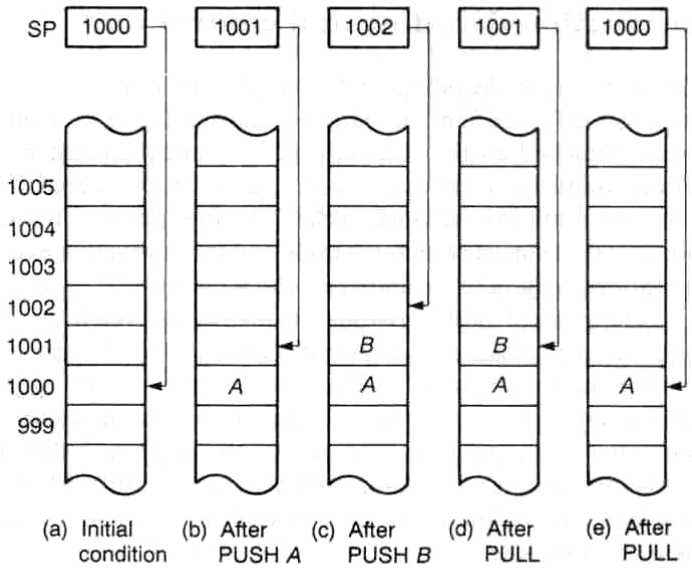



**Figure 12.9** Pushing and pulling information with a stack.

Il puntatore dello stack, SP, viene automaticamente incrementato alla posizione superiore ogni volta che qualsiasi dato è inserito.

Viceversa, ogni volta che un dato viene tolto è solo quello che sta più in alto che viene concesso. Automaticamente il puntatore dello stack viene decrementato alla posizione inferiore.

Tecnica LIFO: Last-In-First-Out

PUSH: Scrittura Dato

PULL: Lettura dato

# Il servizio ai sottoprogrammi: lo stack 2

La semplicità dello stack ha però un'utilità fondamentale. Step-by-step la procedura deve funzionare così:

**Metodo 1:**

1. Memorizzo l'indirizzo del PC nello stack;
2. Chiamo il sottoprogramma inserendo nel PC l'indirizzo del sottoprogramma;
3. Nel sottoprogramma il valore iniziale di tutti i registri che intendo usare li impilo nello stack;
4. Compio le operazioni nel sottoprogramma fornendo i risultati nelle celle di memoria opportune;
5. Prima di ritornare al programma principale ripristino il valore dei registri al valore iniziale;
6. Re-imposto il PC al valore presente nello stack;
7. Ritorno al Programma dalla istruzione successiva a quella di chiamata.

**Metodo 2:**

1. Impilo tutti i registri che sto usando nello stack;
2. Memorizzo l'indirizzo del PC nello stack;
3. Chiamo il sottoprogramma inserendo nel PC l'indirizzo del sottoprogramma;
4. Compio le operazioni nel sottoprogramma fornendo i risultati nelle celle di memoria opportune;
5. Re-imposto il PC al valore presente nello stack;
6. Ritorno al Programma dalla istruzione successiva a quella di chiamata;
7. Ripristino il valore dei registri al valore iniziale.

Il fatto fondamentale è che posso annidare molte chiamate, l'unico limite è la profondità dello stack.

# Il servizio ai sottoprogrammi: lo stack 3

**Osservazione**:

Nella chiamata di un sottoprogramma il **PC** è sempre aggiornato all'indirizzo del sottoprogramma e deve essere ripristinato al ritorno dalla chiamata.

Per questa ragione il trasferimento del **PC** nello stack è in genere eseguito in modo sistematico ed automatico dal sistema di sviluppo.

In genere l'azione di impilare il PC nello stack è individuata dall'uso dell'istruzione **RET** (ritorno da sottoprogramma) o **RETI** (ritorno da interrupt, vedi poi).

Non così viene fatto in genere per i registri, che devono essere gestiti dal progettista. A questo esiste un'eccezione con gli **ARM**, per i quali vedremo in dettaglio.

# La comunicazione con il mondo esterno del micro

Un aspetto fondamentale nell'utilizzo di un micro riguarda la sua capacità di comunicare con il mondo esterno. I rapporti col mondo esterno sono in genere di tipo digitale, ma in alcuni casi possono essere anche di tipo analogico, in quei dispositivi in cui si hanno ADC (Analog to Digital Converter) o DAC (Digital to Analog Converter) contemplati nel micro stesso.

I vari dispositivi sono connessi a dei pin che sono raggruppati in genere in gruppi da 8, o meglio, in gruppi che riflettono il parallelismo del micro. Ogni gruppo è denominato porta. A seconda della complessità del micro si possono avere più porte. Inoltre ogni porta può svolgere più funzioni in modo alternativo

Nella stragrande maggioranza dei casi la porta è gestita come se fosse un registro, utilità che viene denominata "memory mapped input/output". Questo metodo risulta molto comodo perché consente di trattare la porta alla stessa stregua di una qualsiasi cella di memoria: operazioni matematiche di lettura dello stato, ecc. Lo svantaggio è che la porta va indirizzata anche come un registro: si è forzati ad usare il parallelismo degli indirizzi delle celle di memoria anche per selezionare 1 tra 3 o 4 registri presenti.

L'alternativa, sfruttata in alcuni dispositivi, è quella di utilizzare indirizzi speciali per la selezione delle porte.

Ogni porta può disporre di linee di ingresso e linee di uscita: ogni linea di una porta deve potere avere la direzione impostabile.

Associato ad ogni porta vi è un registro, DDR (Data Direction register) che individua lo stato di ogni linea della porta: ingresso o uscita.

Non è escluso che alcune porte possano essere usate in una sola direzione e che alcuni bit di una porta abbiano delle funzioni prestabilite (es. comunicazione seriale).

# Il servizio degli interrupt 1

C'e' una proprietà fondamentale che riguarda la comunicazione tra il micro ed il mondo esterno: la possibilità che un evento esterno possa modificare il corso del programma del micro o necessiti di essere servito con delle azioni opportune.

Esistono diverse modalità in cui questo meccanismo di gestione può essere svolto.

Il caso più semplice si ha quando le unità esterne da gestire sono interrogate dal micro quando lo decide lui, in punti ben precisi del programma che è in esecuzione.

Una situazione più frequente si ha quando il programma compie un "polling", ovvero in modo regolare interroga la periferica e compie delle azioni se ravvisa dei cambiamenti di stato. In una tale situazione può essere utile l'ausilio di un registro addizionale, lo"status register" i cui bit, o "flag" vengono impostati ad 1 quando la periferica ha cambiato stato e vengono azzerati quando il micro legge il registro.

Comunque sia il polling può avvenire secondo 2 modalità:

1) quando interrogo la periferia non faccio niente fino a che non cambia stato oppure;
2) vado avanti a svolgere altre funzioni.

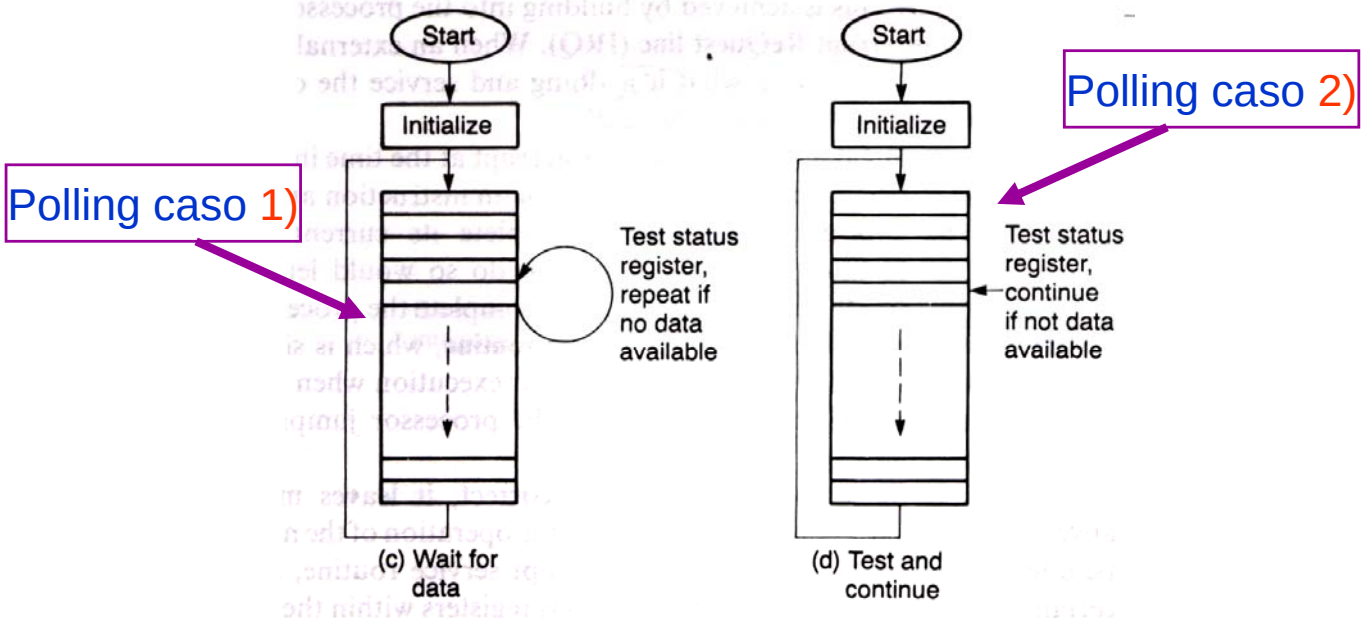



**Figure 12.21** Polling of I/O devices.

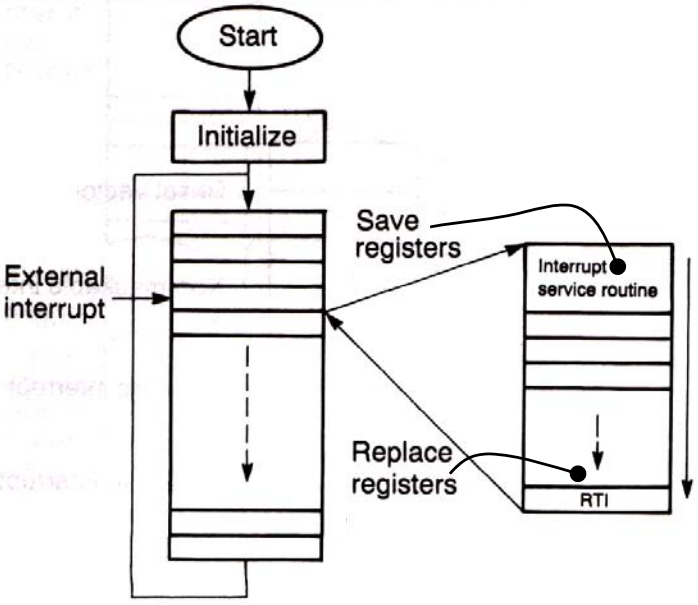
# Il servizio degli interrupt 2

Il polling diventa una situazione dispersiva quando il numero di periferiche da interrogare è elevato.

Si può ricorrere allora all'interrupt. Il micro mette a disposizione un certo numero di linee di ingresso dove le periferiche possono mandare la propria richiesta di interruzione semplicemente cambiando lo stato della linea. Le periferiche facenti capo ad una particolare linea saranno più o meno omogenee.

Alle linee può venire associata una priorità: in caso di contemporaneità nella richiesta viene servita la linea a priorità più elevata.

Se una linea a priorità elevata è in fase di servizio viene ignorata l'eventuale richiesta di una linea a priorità inferiore.

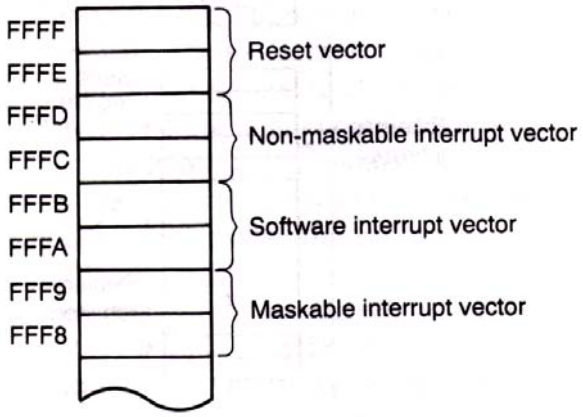



Quando si verifica una situazione di interrupt il micro lancia automaticamente un sottoprogramma il cui indirizzo di partenza in memoria è prestabilito. Ovviamente la procedura dello stack va eseguita.

Nel sottoprogramma di interruzione si prenderà in considerazione la verifica di quale periferica del gruppo in considerazione andrà servita.

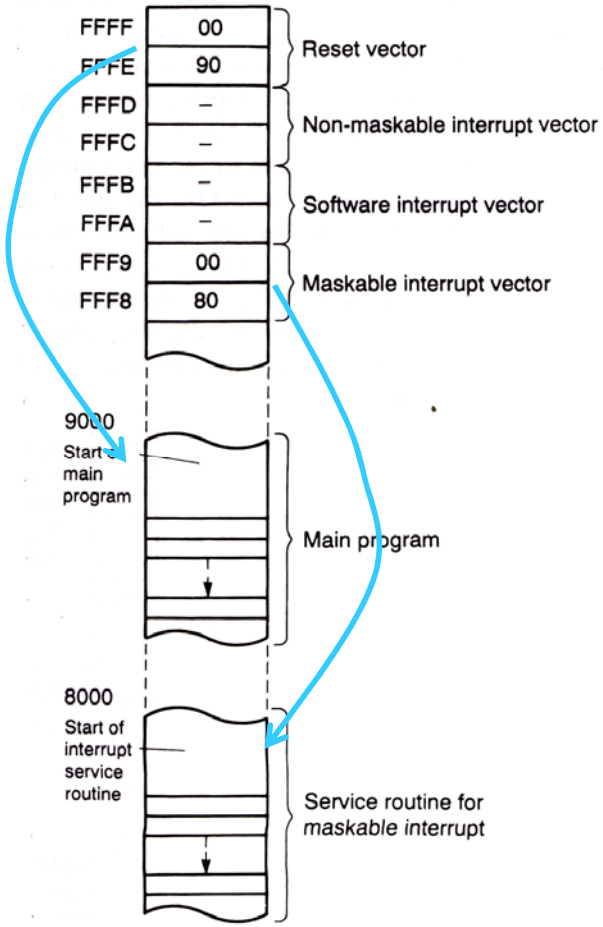
C'è una particolare linea di interruzione che è sempre presente: il **reset**. Ogni micro possiede il pin di reset. Quando il pin di reset viene eccitato, o quando il micro viene acceso, il PC viene impostato al valore dell'indirizzo di partenza del programma, alla cella prestabilita. Il reset ha ovviamente la priorità più elevata.

# Il servizio degli interrupt 3

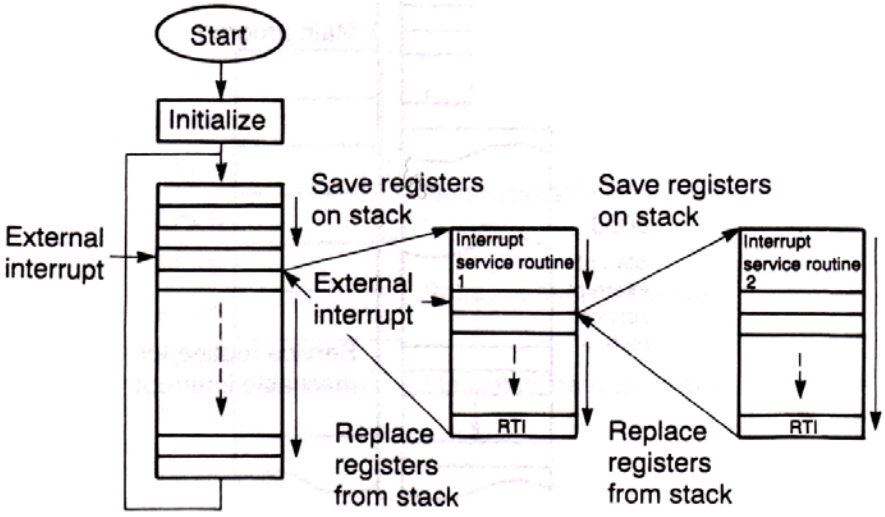



Quando l'interruzione arriva il PC viene fatto saltare alla posizione di memoria, o vettore, dedicata a quella interruzione.

L'interruzione può, eventualmente, essere abilitata ad essere servita o meno, può essere mascherata.

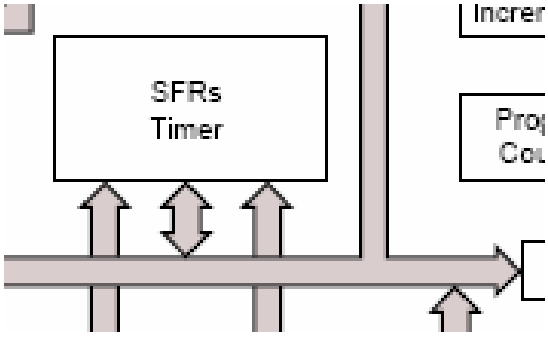



Il concetto è semplicemente che il verificarsi della interrupt abilitata, fa saltare, il PC ad un indirizzo prestabilito della memoria programmi. Lì si metterà un salto alla posizione iniziale del sottoprogramma corrispondente.




Nel caso non si definisse un livello di priorità le interruzioni verrebbero gestite in sequenza, annidando i sottoprogrammi corrispondenti.

# Timer




La gestione delle risorse del micro spesso necessita di compiere azioni secondo una temporizzazione ben precisa.

Per questo è sempre presente in ogni micro almeno un registro che funziona da timer.

Quando abilitato, il valore del registro viene incrementato di uno ad ogni colpo di clock o suoi multipli.

Esistono diverse modalità di funzionamento. Per esempio il timer può essere messo nelle condizioni di ripartire quando arriva al fondo scala, può essere impostato un offset iniziale, ecc.

Cosa importante: quando “smascherato” può generare un’interruzione SW al micro.

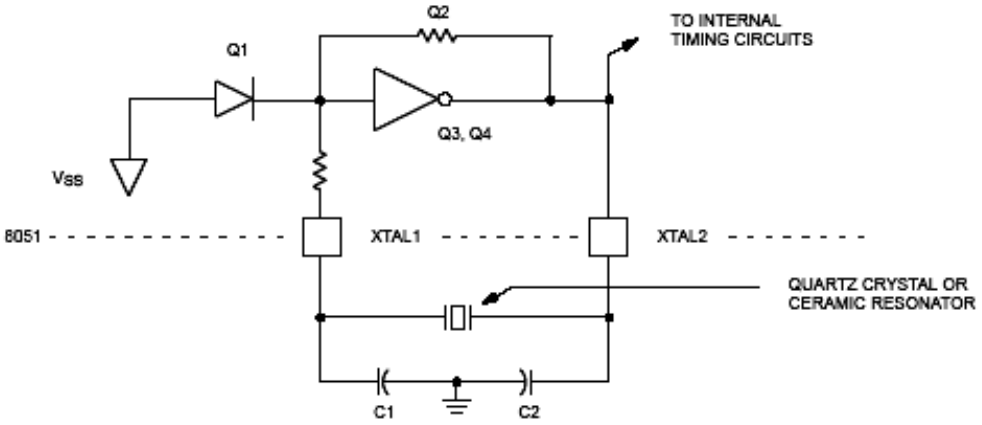
A seconda del micro, più di un timer possono essere contemplati aventi un numero più o meno elevato di modalità di funzionamento.

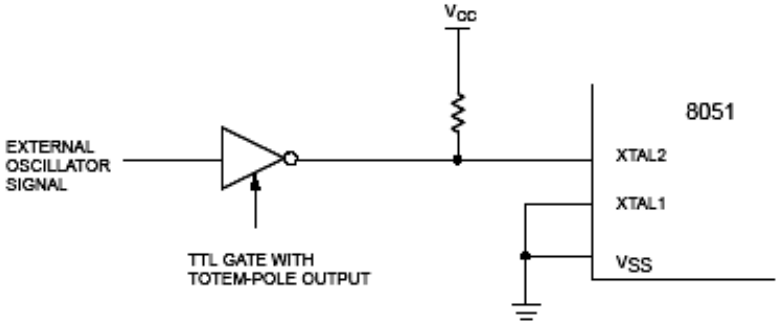
Un esempio di utilizzo del timer è nella trasmissione sia seriale che parallela, dove spesso occorre rispettare intervalli di tempo di attesa prestabiliti tra un dato, o bit, e l’altro.

# Clock

Il clock ha un aspetto molto importante per la vita di un micro. Esistono situazioni dove il clock deve mantenersi molto stabile nel tempo ed in temperatura. Si consideri ad esempio la trasmissione seriale asincrona. Altre situazioni sono meno stringenti.

Ci sono micro che hanno già integrato un generatore di clock e non necessitano della presenza di ulteriori componenti esterni. Questi micro hanno in genere una basa emissione di disturbi EMI. I micro che hanno il clock già implementato ammettono comunque la connessione e l'utilizzo di un clock esterno, se l'applicazione richiedesse una frequenza di lavoro particolare, o sincronismo con altri dispositivi.




Il clock può essere fornito secondo 2 modalità.

1) Aggiungendo un cristallo risuonatone (ed anche, in genere, 2 condensatori di stabilizzazione) connesso a 2 piedini appositi.

2) Fornendo direttamente il segnale di clock ad uno dei 2 piedini adibiti alla gestione del clock.

ATTENZIONE: per ottenere clock di velocità elevata i cristalli risuonatori non bastano. Si usa la tecnica PLL (Phase Locked Loop), che vedremo più avanti nel corso.

# Struttura del microcontrollore 9

La struttura completa del micro basato sul “core” 8051 appare come qui sotto:

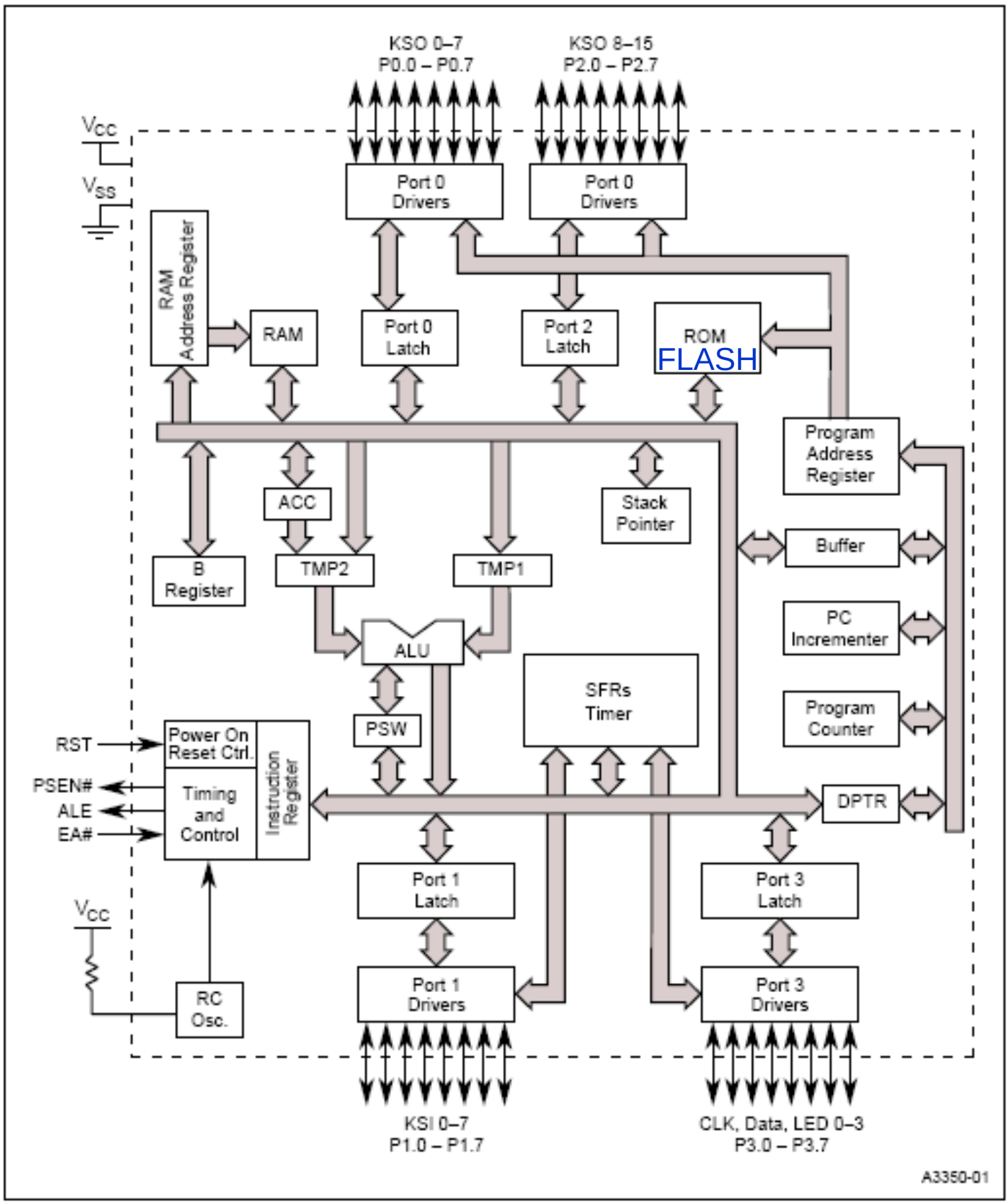



Figure 2. 83C51KB Block Diagram

# Esempio di ciclo con Motorola M68HC08

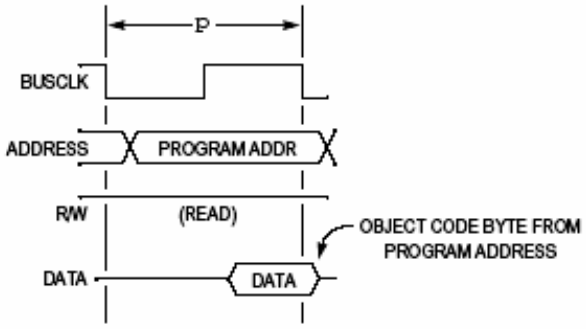



**Figure 1. Timing Waveforms for a Program Fetch (p) Cycle**

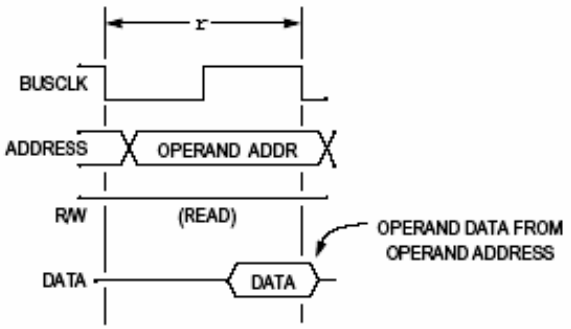



**Figure 2. Timing Waveforms for a Data Read (r) Cycle**

Il fetch, il read ed il write sono eseguite in un solo colpo di clock.

L'istruzione BCLR Bit_0, port0 imposta uno 0 nel bit '0' della porta B (in logica negata).

La sequenza è prwp, ovvero 4 colpi di clock.

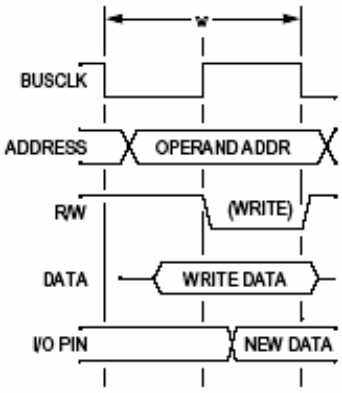



**Figure 3. Timing Waveforms for a Write (w) Cycle**

Cella 8310: op code 11 (già nell'Instr. Register);
p: Cella 8311: l'indirizzo basso della porta è caricato;
r: Viene letto il contenuto della porta;
w: Viene scritta la porta con il Bit_0 aggiornato;
p: fetch della istr. successiva, un salto.

```
8310 11 01      loop:   BCLR  Bit_0,PortB  ;[4] drive PTB0 low
8312 20 FC              BRA   loop         ;[3] repeat
```

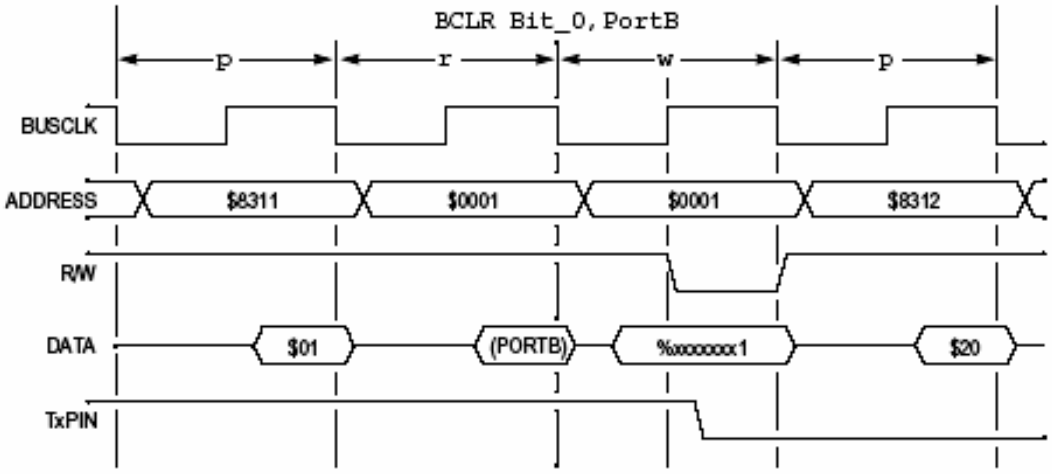



**Figure 8. Timing Details for a BCLR Instruction**

# La rappresentazione numerica 1

I numeri possono essere rappresentati in diversi modi che determinano maggiore o minore precisione. Ovviamente più precisa è la rappresentazione, più costosa è la implementazione.

Il modo più semplice è la rappresentazione per interi senza segno. Questo è sostanzialmente il modo di operare degli ADC e DAC, per cui è la più simile a quello che succede nel mondo reale.

Semplicemente ad ogni cifra viene associato un numero intero con peso corrispondente alla propria posizione.

Ad es. con 8 bit si possono rappresentare fino a 256, con 16 bit fino a 65536, con 32 bit fino a $4.29\ 10^9$.

Dalla rappresentazione mediante intero senza segno si passa alla rappresentazione degli interi con segno. In questo caso a parità di numero di cifre si perde un bit per il segno, ovvero il range rimane inalterato ma diviso per 2. Con 8 bit si rappresenta da –128 a +127, con 16 bit da –32768 a +32767, con 32 bit $\pm 2.147\ 10^9$.

La rappresentazione più completa è quella così detta “floating point” dove si riescono a rappresentare numeri in virgola mobile, con segno.

E’ concepita nella forma $f.fff \times 2^a$ con a=esponente, ed f.fff=mantissa (ovviamente il numero di cifre f dipende dal numero di bit a disposizione).

Es. con 32 bit si riesce a rappresentare $\pm 3.4 \times 10^{\pm 38}$.

# La rappresentazione numerica 2

La rappresentazione “floating point” è gestita solo dai micro più sofisticati, i DSP. Ha un costo in termini di HW / tempo di esecuzione. I DSP più veloci riescono a compiere operazioni su dati decimali anche in un singolo colpo di clock utilizzando risorse HW appositamente dedicate.

La complessità risiede nel fatto che i dati devono essere trasformati prima di essere elaborati. Per es. se dobbiamo fare la somma tra 2 numeri dobbiamo trasformare i 2 numeri in modo che abbiano lo stesso esponente, poi possiamo sommare le mantisse.

Il prodotto tra 2 numeri si fa moltiplicando tra loro le mantisse e sommando gli esponenti.

Alla fine comunque le operazioni vengono svolte su numeri interi con segno.

Es. $4.38\times10^{15} + 2.15\times10^{13}$=(dopo una doppia rotazione) = $(438+2.15)\times10^{13}=440.15\times2^{13}$.

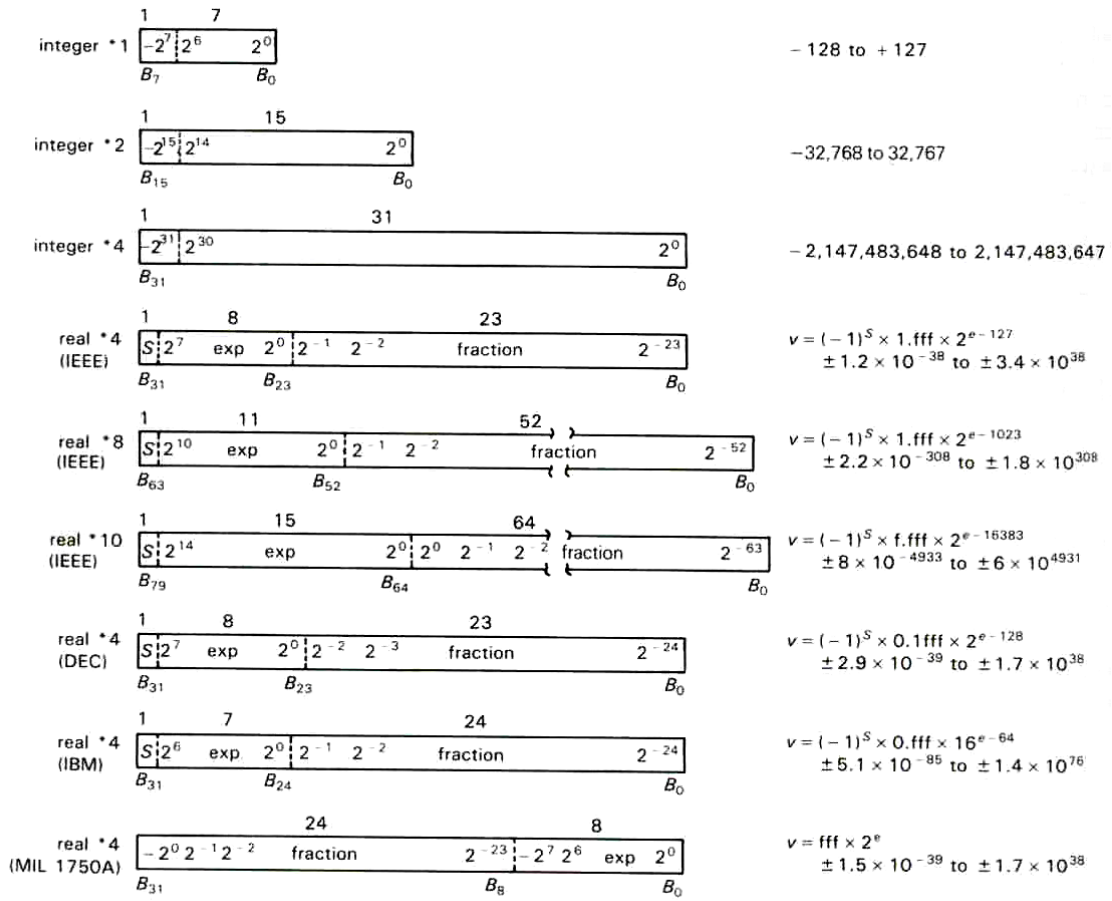



Figure 10.24. Number formats.

# La rappresentazione numerica 3: la somma e la differenza

La rappresentazione dei numeri con segno in codice binario viene svolta in modo da semplificare le operazioni HW da svolgere.

La rappresentazione più proficua è quella così detta in complemento a 2, che consente di trasformare la differenza tra 2 numeri nella somma.

Concetto in termini decimali:

Supponiamo di disporre di n cifre decimali. Il numero massimo rappresentabile è pertanto $10^n$-1. La differenza tra 2 numeri A e B si può scrivere come:

$$A-B=A-B+(10^n-1)-(10^n-1)$$

$$=A+[(10^n-1-B)+1]-10^n$$

Una volta costruito il numero entro parentesi quadre l'operazione è una somma della quale basta non considerare il riporto, dato dal termine $-10^n$.
Quando si rappresentano cifre binarie il numero entro parentesi ha una espressione molto comoda:

$$2^n-1-B+1=\overline{B}+1$$

Perciò:

$$A-B=A+\overline{B}+1,\ \text{con omesso il riporto generato}$$

Es. n=5 bit.

$+13+$ $\quad 13d=01101b$

$-10=$ $\quad 10d=01010b \Rightarrow -10d=10110b$

$3$

$01101b+$
$10110b=$
$1\,00011b$

$-7+$ $\quad 7d=00111b \Rightarrow -7d=11001b$

$-8=$ $\quad 8d=01000b \Rightarrow -8d=11000b$

$-15$

$11001b+$
$11000b=$
$1\,10001b$

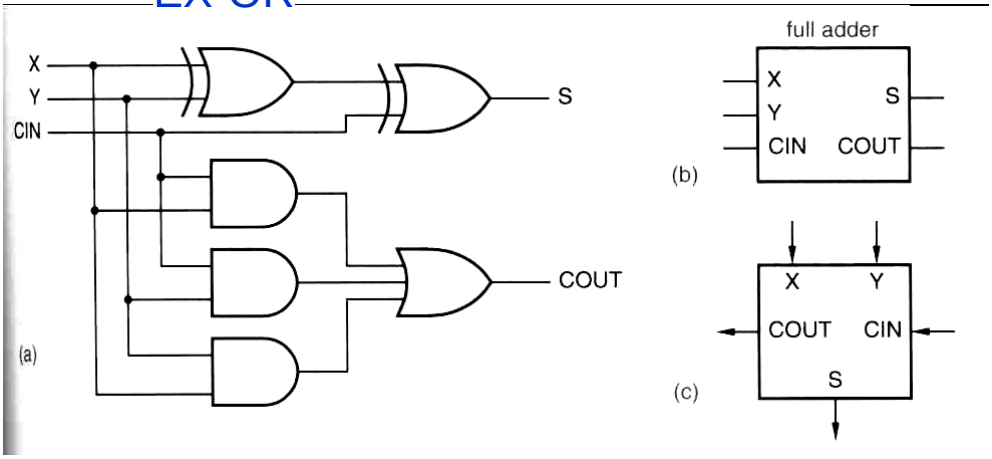
Ma:

$$10001b \xrightarrow{\text{Cambio segno}} 01110b+1b=01111b=15d \Rightarrow$$

$$10001b \leftrightarrow -15d$$

# La rappresentazione numerica 4: la somma e la differenza

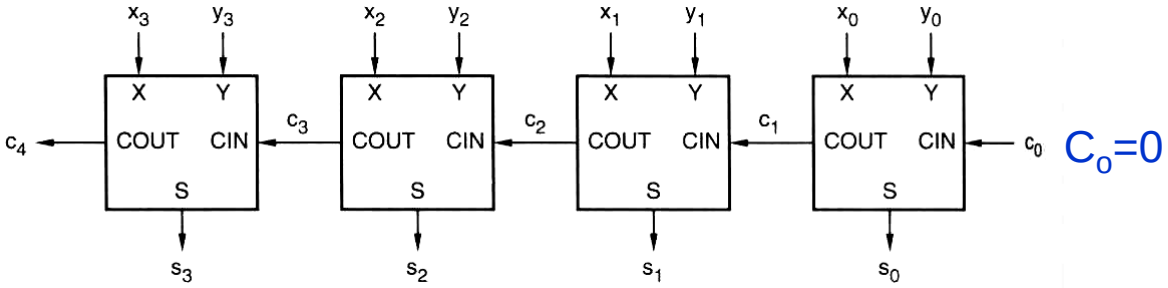
La somma del singolo bit si realizza con combinazioni di EX-OR:




**FULL-ADDER**

Ponendo in cascata più FULL-ADDER con i riporti connessi anche in cascata si ottiene la somma a più bit.

Questa configurazione non è la più veloce perché nelle condizioni peggiori il riporto dal primo bit deve arrivare all'ultimo.




La sottrazione X-Y si può realizzare sfruttando la somma se si ha cura di compiere la trasformazione:

$$X + \overline{Y}$$

In aggiunta il riporto di ingresso del sommatore va posto =1:

$$C_0 = 1$$

# La rappresentazione numerica 5: la moltiplicazione e la divisione

```
(11) 01011×
(12) 01100 =
      00000
     00000
    01011
   01011
1000100  (132)
```

Nella moltiplicazione è più conveniente calcolare le semisomme parziali:

```
(11) 01011×
 (9) 01001 =
      00000
     01011
    001011
   00000
   0001011
  00000
  00001011
  01011
  01100011        (99)
```

Per la moltiplicazione di numeri negativi si usa il complemento a 2:

```
(-7) 1001×        moltiplicando
(-3) 1101 =       moltiplicatore
      00000
     11001
    111001
    00000
   1111001
   11001
  11011101
  00111       (L'ultimo numero è complementato se il moltiplicatore è < 0)
  00010101    (21)
```

Occorre aggiungere una colonna con un 1 se il moltiplicando è negativo: la moltiplicazione aggiunge una cifra, per cui si perderebbe l'informazione sul segno se non si aggiungesse l'1.

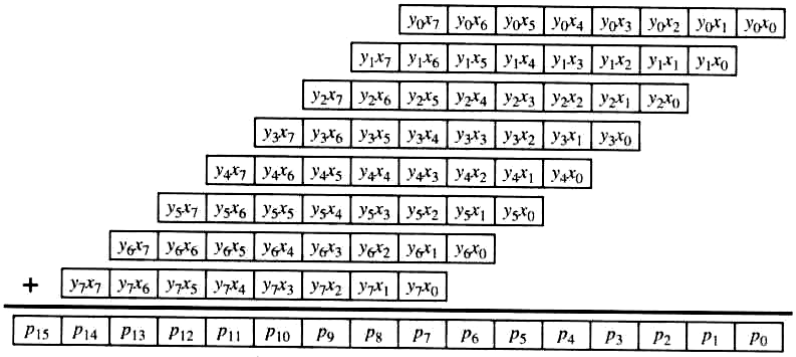
Nell'ultima moltiplicazione, quella con il bit + significativo, occorre invertire in modulo 2 il moltiplicando, se il moltiplicatore è negativo.

# La rappresentazione numerica 6: la moltiplicazione e la divisione

```
      19                  10011   quotient
11 )217          1011  )11011001  dividend
      11                    1011   shifted divisor
     107                    0101   reduced dividend
      99                    0000   shifted divisor
       8                    1010   reduced dividend
                           0000   shifted divisor
                          10100   reduced dividend
                           1011   shifted divisor
                          10011   reduced dividend
                           1011   shifted divisor
                           1000   remainder
```

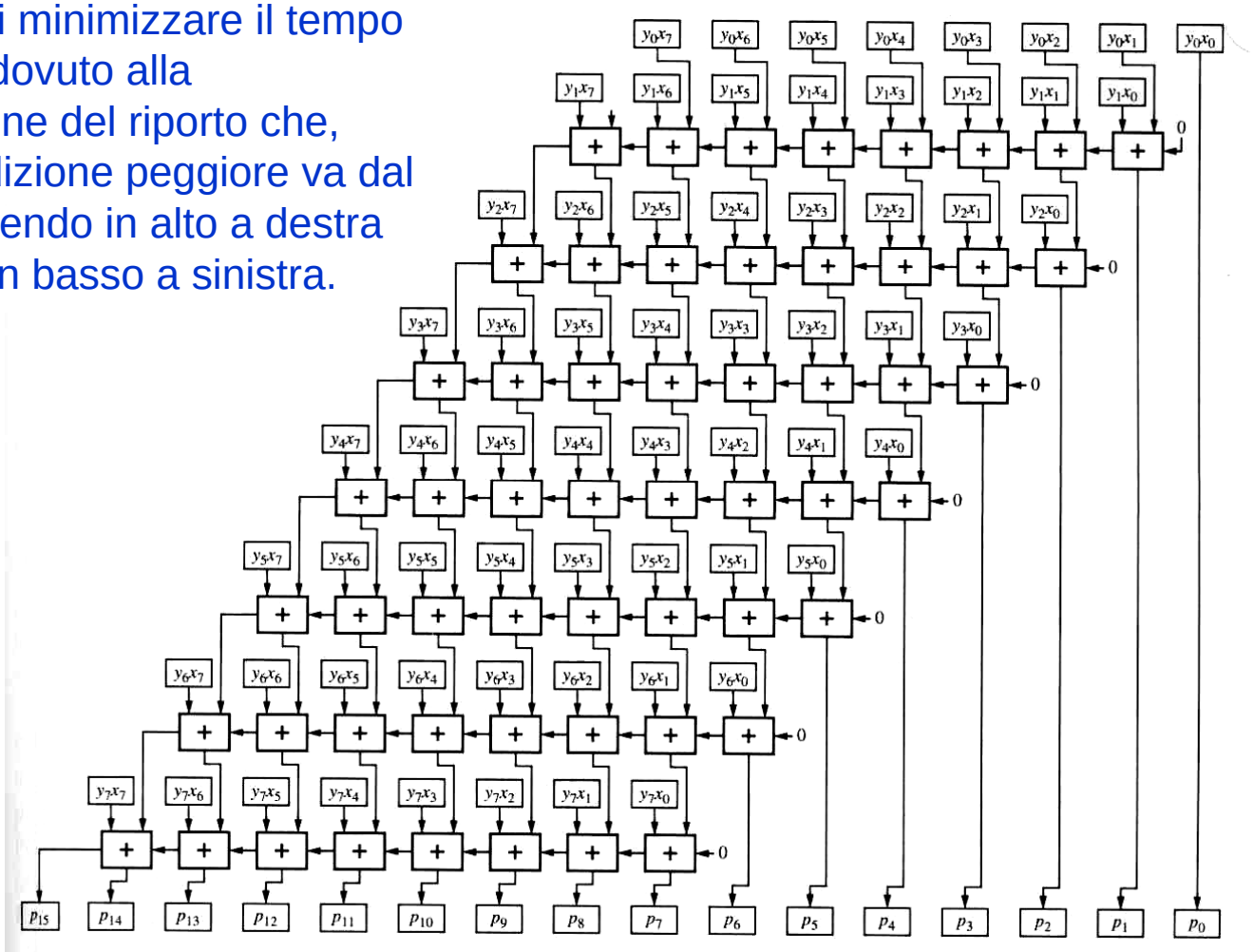
La divisione si riconduce a somme e prodotti con algoritmi più o meno sofisticati.

Al solito la velocità va a scapito della complessità circuitale e viceversa.




Alla fine il costo HW da sostenere per ottenere una moltiplicazione efficiente e veloce si deve pagare.

Esistono poi soluzioni che cercano di minimizzare il tempo di ritardo dovuto alla trasmissione del riporto che, nella condizione peggiore va dal primo addendo in alto a destra all'ultimo in basso a sinistra.



# Il microcontrollore Intel 8051 1

L'8051 è un micro avente una struttura HW abbastanza semplice che ha una grande diffusione. La sua fama è anche dovuta al fatto che è stato uno dei primi micro apparsi sul mercato.

Sebbene molto diffuso l'8051 non ha certamente la struttura HW più veloce nell'esecuzione delle istruzioni. La macchina standard impiega 12 colpi di clock, in genere, per compiere un'istruzione.

Una delle ragioni sta nel fatto che la macchina è basata su di una struttura così detta μ-code.

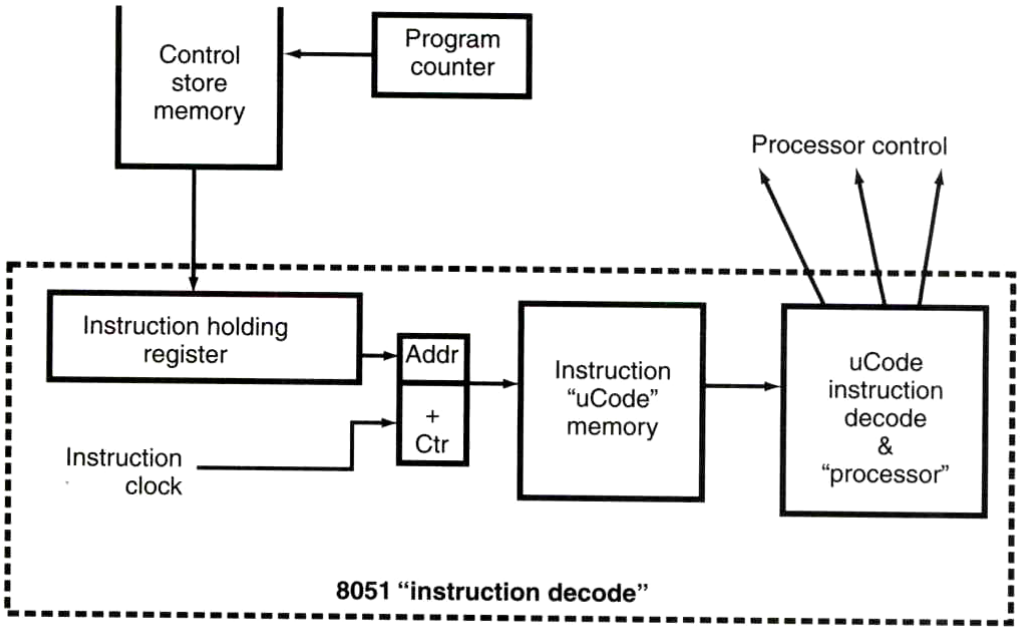



**FIGURE 2-15** 8051's microcoded processor.

Ogni istruzione viene eseguita come sottoprogramma. Questo approccio consente una certa semplicità progettuale. Tuttavia questo vantaggio va a scapito della velocità di esecuzione.

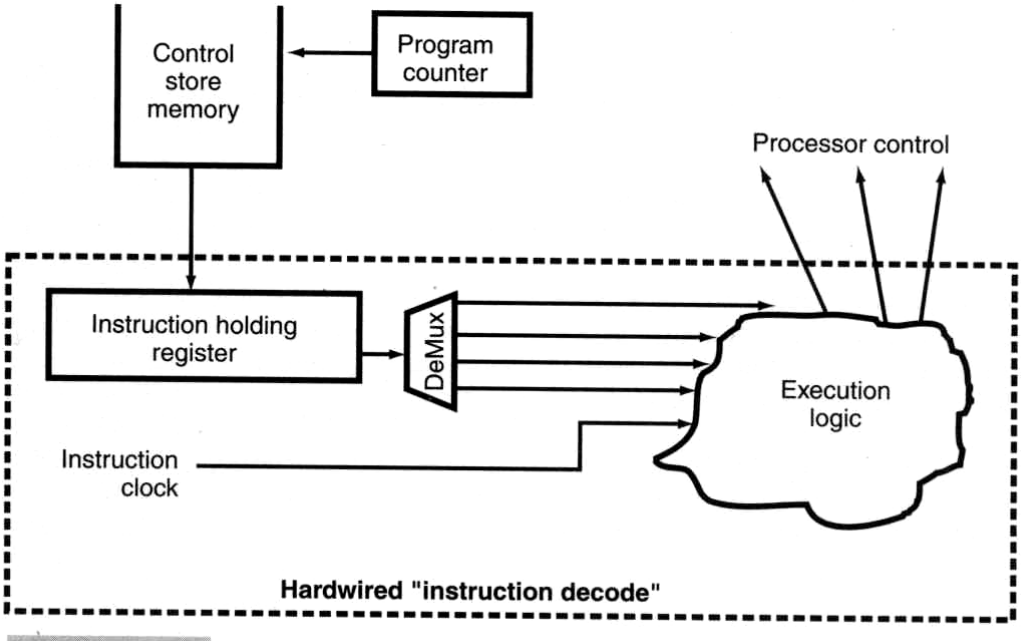



**FIGURE 2-16** Hardwired instruction processor.

Questa metodologia è in contrasto con quella così detta hardwired in cui l'istruzione è usata per comandare direttamente le varie unità preposte attraverso una decodifica logica.

La hardwired è più veloce, ma più complessa dal punto di vista progettuale.

# Il microcontrollore Intel 8051 2

Le unità principali dell'8051 ricalcano quelle che si possono visualizzare in qualsiasi altro micro.

Possiede una memoria dati e programmi interne. Nella maggiore parte dei prodotti la RAM interna è limitata a 256 byte.

La memoria programmi posta all'interno varia da modello a modello, da qualche kbyte a 64 kbyte e più.

Il parallelismo è a 8 bit con un indirizzamento a 16 bit, ovvero 64 Kbyte di memoria indirizzabile, a meno di fare uso di soluzioni dedicate.

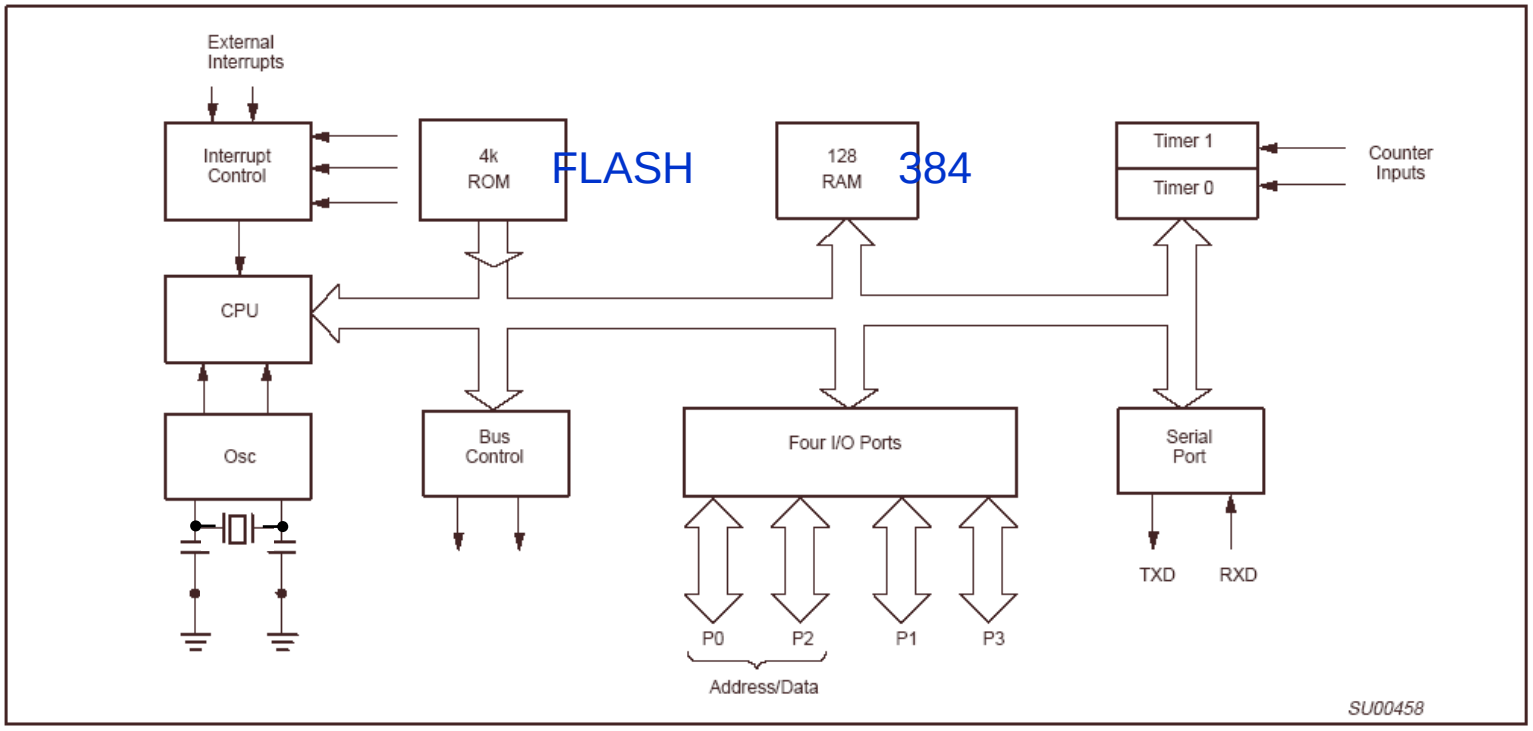



**Figure 1. 80C51 Block Diagram**

Nella soluzione più completa si hanno 4 porte di I/O. Però in caso si necessiti del supporto della memoria esterna le porte P0 e P2 vengono dedicate alla sola gestione della memoria. Avendo a disposizione solo 16 linee da dedicare all'indirizzamento si deve ricorrere alla soluzione avente latch esterno.

E' possibile che siano incluse delle porte, con 2 o più linee, seriali.

Vi è una unità adibita alla generazione dei segnali di controllo.

Vi sono 2 o più timer.

Vi è un'unità che serve a generare il clock, l'ALU e l'unità di interupt.

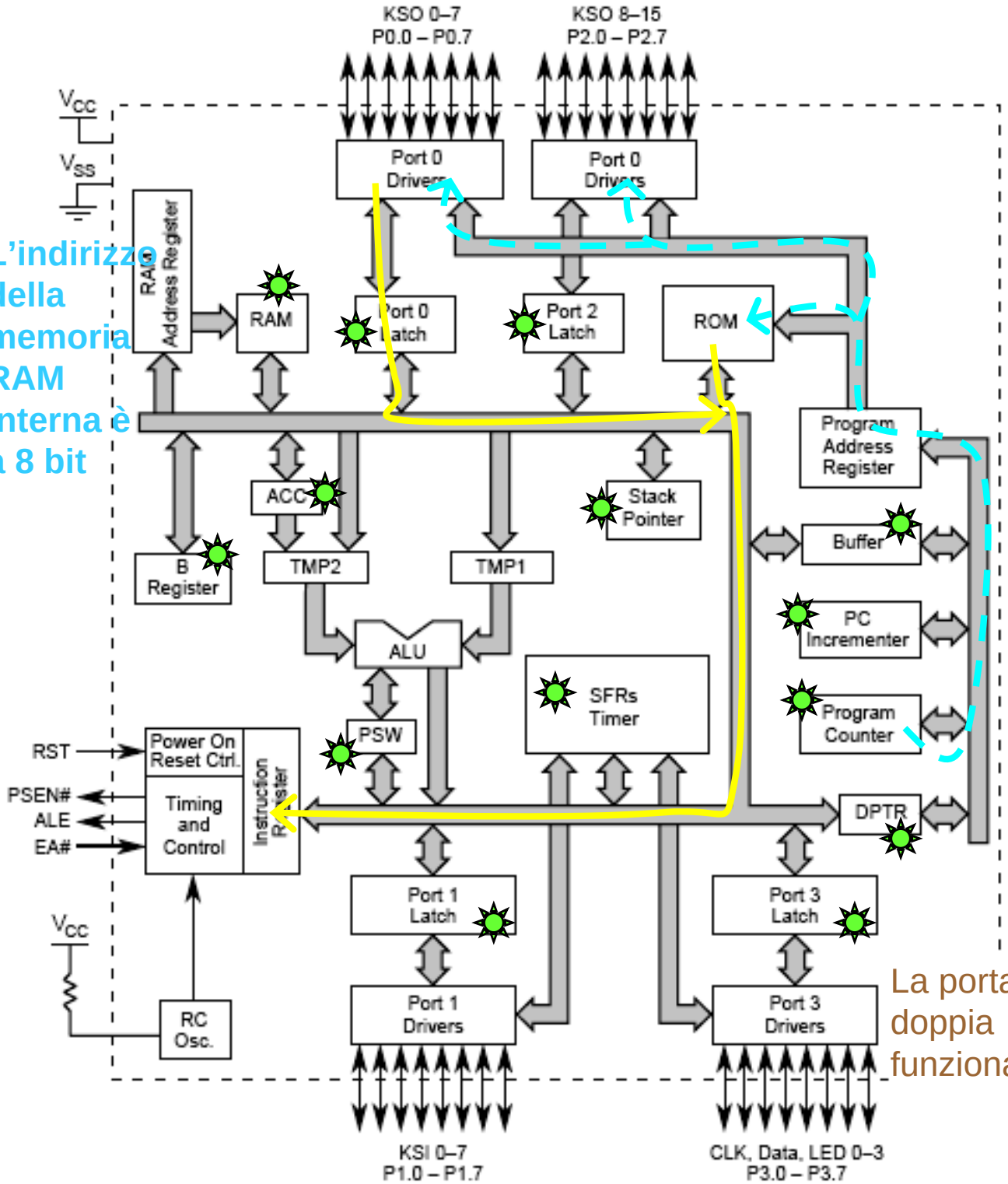
# Il microcontrollore Intel 8051 3

= unità facenti tutte parti della RAM dati. A questi registri sono assegnati dei nomi, ma potrebbero essere indirizzati con l'indirizzo della loro cella.

= possibile percorso per l'indirizzo dell'istruzione.

= possibile percorso per l'istruzione di cui si è eseguito il fetch.

L'indirizzo della memoria RAM interna è a 8 bit




La porta 3 ha una doppia funzionalità

# L'organizzazione della memoria dell'Intel 8051 1

Un aspetto fondamentale nella comprensione del funzionamento di un micro è la organizzazione della memoria ed i conseguenti tipi di indirizzamento. Nell'8051 abbiamo l'opportunità di disporre sia della memoria dati esterna che interna. Così pure accade per la memoria programmi.

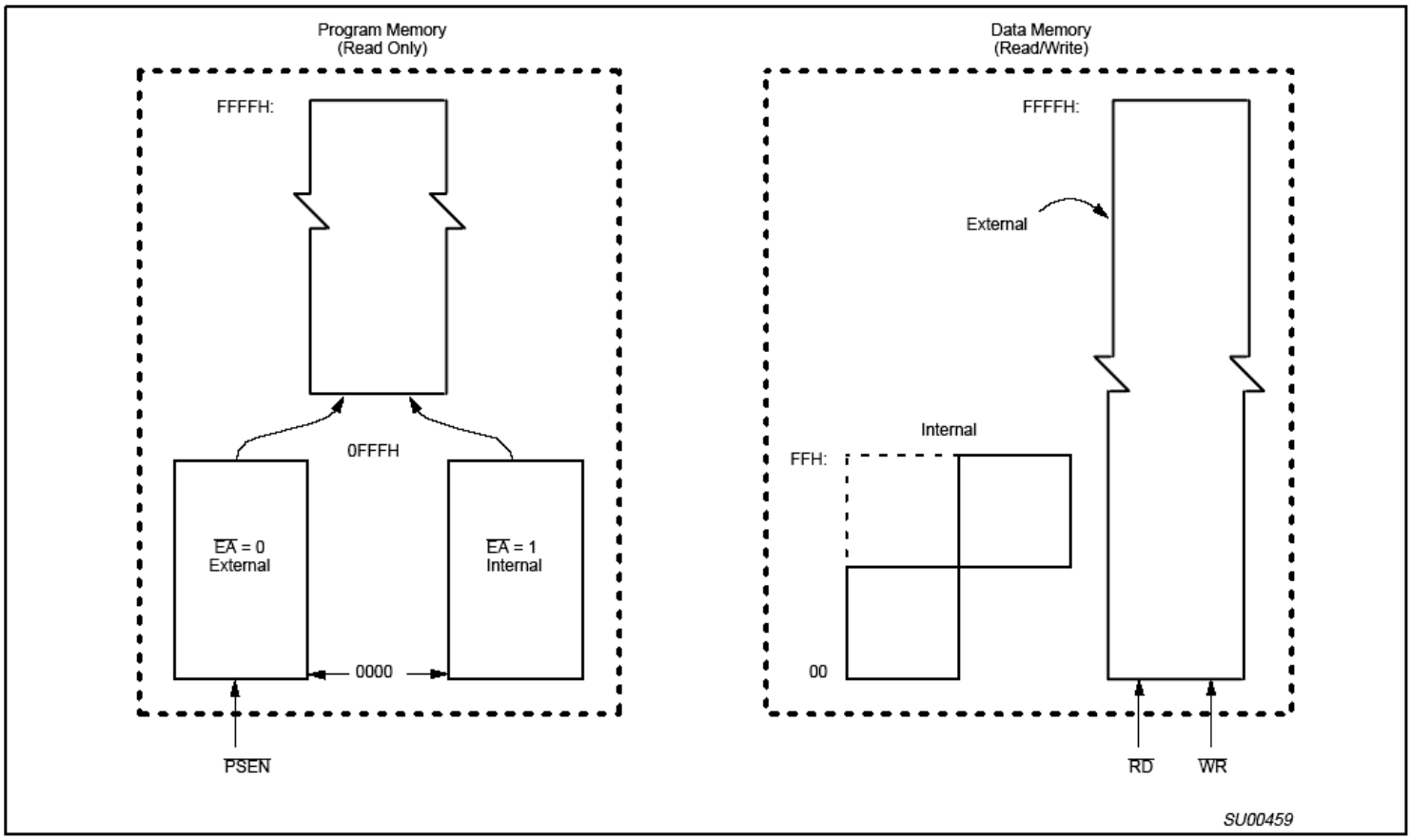



**Figure 2. 80C51 Memory Structure**

La memoria programmi può seguire 2 modalità. La prima è di disporre di una parte di memoria programmi interna ed una esterna, postagli sopra. Il PC passa automaticamente alla memoria esterna sopra l'indirizzo ultimo interno, 0FFFh nell'esempio.

Esiste il pin EA che consente, quando pilotato basso, di usufruire solo della memoria esterna. Questa adozione è importante perché cambia anche il posizionamento dei vettori di interrupt.

Anche la memoria dati può avere parte residente esternamente. In questo caso i dati dall'esterno possono essere manipolati con l'istruzione MOVX.

La RAM interna è organizzata in strutture distinguibili per la loro indirizzabilità.

# L'organizzazione della memoria dell'Intel 8051 2

La RAM è divisa in 3 blocchi forse perché è incrementata di dimensioni nel corso del tempo.

Il primo blocco si estende dall'indirizzo 0h all'indirizzo 7Fh (127d).

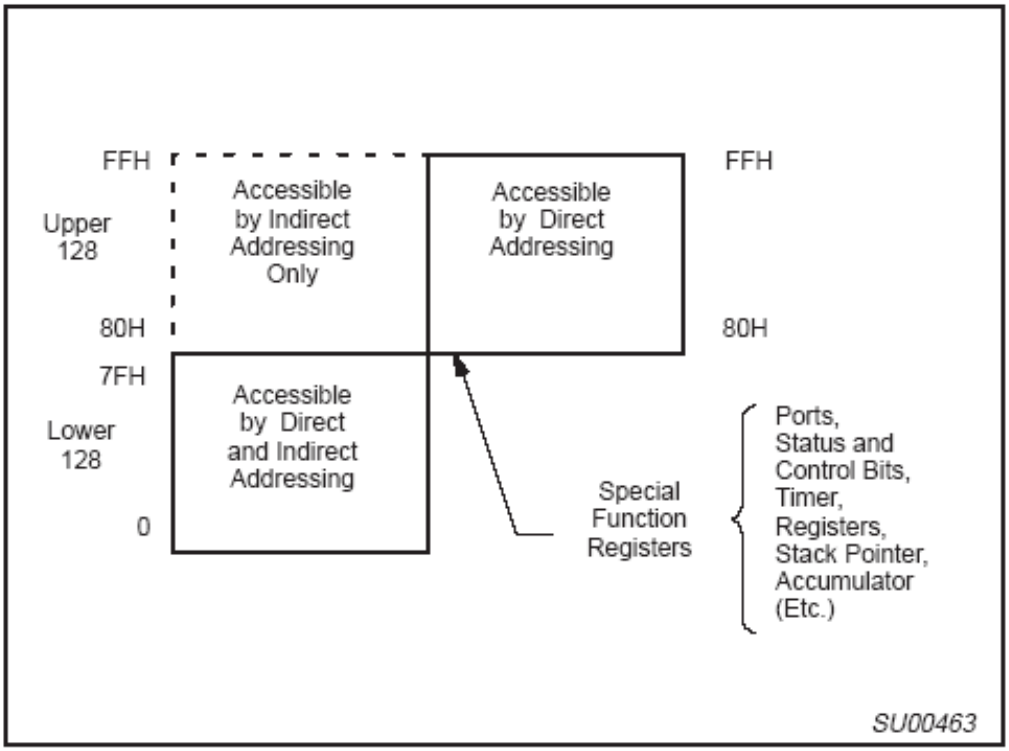



**Figure 6. Internal Data Memory**

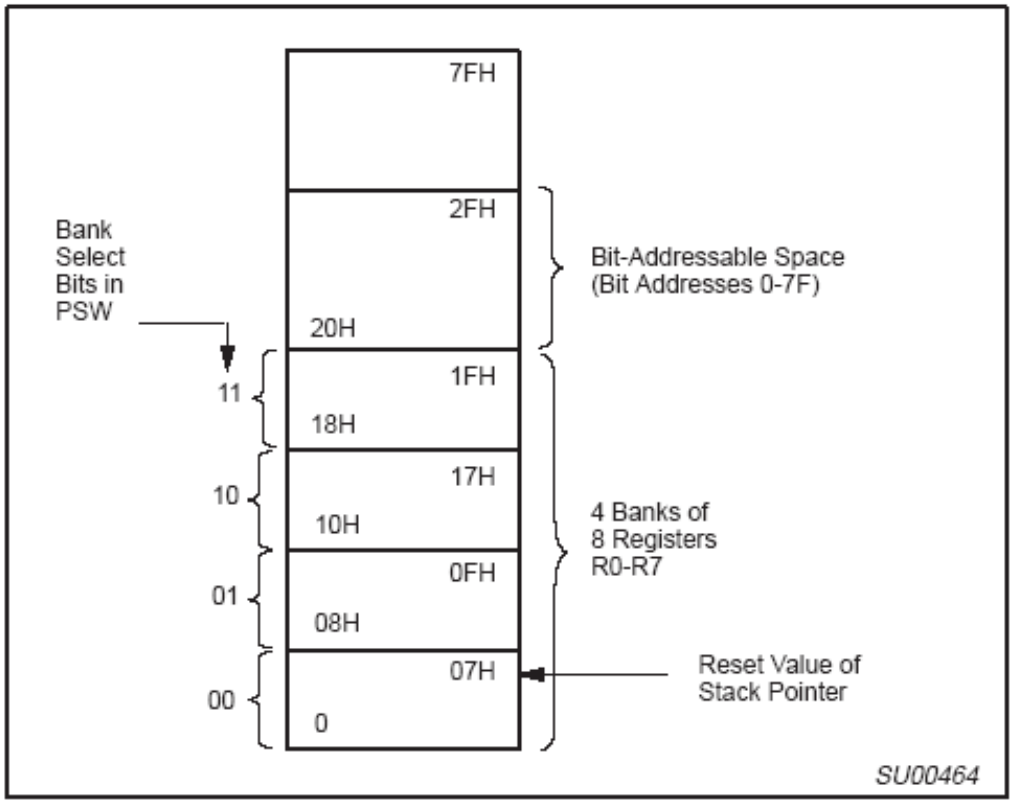



**Figure 7. Lower 128 Bytes of Internal RAM**

I primi 32 byte sono divisi in 4 banchi da 8 byte ognuno.

All'interno di ogni banco le celle possono essere indirizzate come registri, R0-R7.

2 bit nel PSW consentono di selezionare il banco.

In ogni banco i registri R0 ed R1 sono usati anche per determinare l'indirizzamento indiretto.

Da 20h a 2Fh i registri sono liberi e godono della proprietà di essere indirizzabili per bit, oltre che per parola.

Da 30h a 7Fh si hanno altri registri liberi indirizzabili solo per byte.

# L'organizzazione della memoria dell'Intel 8051 3

L'indirizzo interno della memoria RAM è a 8 bit. Di principio si potrebbero indirizzare solo 256 byte, a meno di rallentare il processo di movimentazione dei dati aggiungendo un byte di indirizzo. Per arrivare ad avere 384 dati si è usato un trucco che, sostanzialmente, consente di aggiungere un bit virtuale

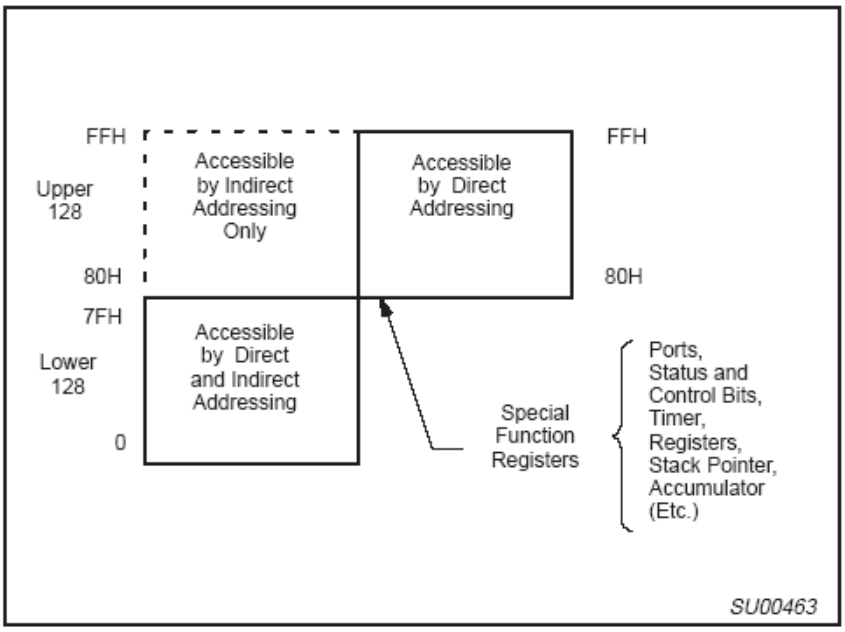
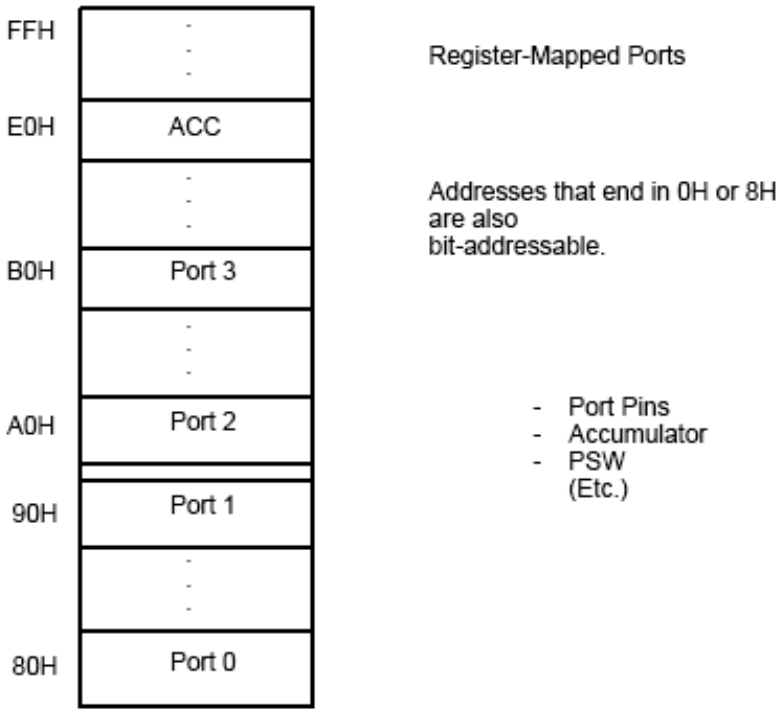



**Figure 6. Internal Data Memory**

Usando l'indirizzamento indiretto, via R0 o R1, si agisce sui 128 byte da 80h a FFh come registri general purpose, non indirizzabili per bit.

Es. MOV A, @R0




Usando un indirizzamento diretto si accede alla zona di memoria contenente gli **SFR** (Special Function registers). Qui c'e' l'accumulatore, i registri per la gestione delle porte, lo stack pointer, ecc. Inoltre molti registri contenuti qui dentro sono indirizzabili per bit.

Es. MOV A,128h

# L'organizzazione della memoria dell'Intel 8051 4

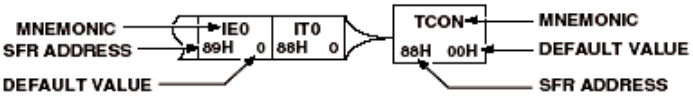
Un esempio di come sia organizzata la memoria che riguarda gli SFR, comprensivo dell'indirizzamento delle celle per bit. Questo micro è l'ADUC832 di Analog. I registri speciali di questo dispositivo sono molti perché ha molte funzioni addizionali rispetto ad un 8051 standard.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ISPI FFH 0 | WCOL FEH 0 | SPE FDH 0 | SPIM FCH 0 | CPOL FBH 0 | CPHA FAH 1 | SPR1 F9H 0 | SPR0 F8H 0 | BITS | SPICON[1] F8H 04H |
| F7H 0 | F6H 0 | F5H 0 | F4H 0 | F3H 0 | F2H 0 | F1H 0 | F0H 0 | BITS | B[1] F0H 00H |
| MDO EFH 0 | MDE EEH 0 | MCO EDH 0 | MDI ECH 0 | I2CM EBH 0 | I2CRS EAH 0 | I2CTX E9H 0 | I2CI E8H 0 | BITS | I2CCON[1] E8H 00H |
| E7H 0 | E6H 0 | E5H 0 | E4H 0 | E3H 0 | E2H 0 | E1H 0 | E0H 0 | BITS | ACC[1] E0H 00H |
| ADCI DFH 0 | DMA DEH 0 | CCONV DDH 0 | SCONV DCH 0 | CS3 DBH 0 | CS2 DAH 0 | CS1 D9H 0 | CS0 D8H 0 | BITS | ADCCON2[1] D8H 00H |
| CY D7H 0 | AC D6H 0 | F0 D5H 0 | RS1 D4H 0 | RS0 D3H 0 | OV D2H 0 | FI D1H 0 | P D0H 0 | BITS | PSW[1] D0H 00H |
| TF2 CFH 0 | EXF2 CEH 0 | RCLK CDH 0 | TCLK CCH 0 | EXEN2 CBH 0 | TR2 CAH 0 | CNT2 C9H 0 | CAP2 C8H 0 | BITS | T2CON[1] C8H 00H |
| PRE3 C7H 0 | PRE2 C6H 0 | PRE1 C5H 0 | PRE0 C4H 1 | WDIR C3H 0 | WDS C2H 0 | WDE C1H 0 | WDWR C0H 0 | BITS | WDCON[1] C0H 10H |
| PSI BFH 0 | PADC BEH 0 | PT2 BDH 0 | PS BCH 0 | PT1 BBH 0 | PX1 BAH 0 | PT0 B9H 0 | PX0 B8H 0 | BITS | IP[1] B8H 00H |
| RD B7H 1 | WR B6H 1 | T1 B5H 1 | T0 B4H 1 | INT1 B3H 1 | INT0 B2H 1 | TxD B1H 1 | RxD B0H 1 | BITS | P3[1] B0H FFH |
| EA AFH 0 | EADC AEH 0 | ET2 ADH 0 | ES ACH 0 | ET1 ABH 0 | EX1 AAH 0 | ET0 A9H 0 | EX0 A8H 0 | BITS | IE[1] A8H 00H |
| A7H 1 | A6H 1 | A5H 1 | A4H 1 | A3H 1 | A2H 1 | A1H 1 | A0H 1 | BITS | P2[1] A0H FFH |
| SM0 9FH 0 | SM1 9EH 0 | SM2 9DH 0 | REN 9CH 0 | TB8 9BH 0 | RB8 9AH 0 | TI 99H 0 | RI 98H 0 | BITS | SCON[1] 98H 00H |
| 97H 1 | 96H 1 | 95H 1 | 94H 1 | 93H 1 | 92H 1 | T2EX 91H 1 | T2 90H 1 | BITS | P1[1,2] 90H FFH |
| TF1 8FH 0 | TR1 8EH 0 | TF0 8DH 0 | TR0 8CH 0 | IE1 8BH 0 | IT1 8AH 0 | IE0 89H 0 | IT0 88H 0 | BITS | TCON[1] 88H 00H |
| 87H 1 | 86H 1 | 85H 1 | 84H 1 | 83H 1 | 82H 1 | 81H 1 | 80H 1 | BITS | P0[1] 80H FFH |

| | | | | | | |
|---|---|---|---|---|---|---|
| SPICON[1] F8H 04H | DAC0L F9H 00H | DAC0H FAH 00H | DAC1L FBH 00H | DAC1H FCH 00H | DACCON FDH 04H | RESERVED | RESERVED |
| B[1] F0H 00H | ADCOFSL[3] F1H 00H | ADCOFSH[3] F2H 20H | ADCGAINL[3] F3H 00H | ADCGAINH[3] F4H 00H | ADCCON3 F5H 00H | RESERVED | SPIDAT F7H 00H |
| I2CCON[1] E8H 00H | RESERVED | RESERVED | RESERVED | RESERVED | RESERVED | RESERVED | ADCCON1 EFH 00H |
| ACC[1] E0H 00H | RESERVED | RESERVED | RESERVED | RESERVED | RESERVED | RESERVED | RESERVED |
| ADCCON2[1] D8H 00H | ADCDATAL D9H 00H | ADCDATAH DAH 00H | RESERVED | RESERVED | RESERVED | RESERVED | PSMCON DFH DEH |
| PSW[1] D0H 00H | RESERVED | DMAL D2H 00H | DMAH D3H 00H | DMAP D4H 00H | RESERVED | RESERVED | PLLCON D7H 53H |
| T2CON[1] C8H 00H | RESERVED | RCAP2L CAH 00H | RCAP2H CBH 00H | TL2 CCH 00H | TH2 CDH 00H | RESERVED | RESERVED |
| WDCON[1] C0H 10H | RESERVED | CHIPID C2H 2XH | RESERVED | RESERVED | RESERVED | EDARL C6H 00H | EDARH C7H 00H |
| IP[1] B8H 00H | ECON B9H 00H | RESERVED | RESERVED | EDATA1 BCH 00H | EDATA2 BDH 00H | EDATA3 BEH 00H | EDATA4 BFH 00H |
| P3[1] B0H FFH | PWM0L B1H 00H | PWM0H B2H 00H | PWM1L B3H 00H | PWM1H B4H 00H | NOT USED | NOT USED | SPH B7H 00H |
| IE[1] A8H 00H | IEIP2 A9H A0H | RESERVED | RESERVED | RESERVED | RESERVED | PWMCON AEH 00H | CFG832 AFH 00H |
| P2[1] A0H FFH | TIMECON A1H 00H | HTHSEC A2H 00H | SEC A3H 00H | MIN A4H 00H | HOUR A5H 00H | INTVAL A6H 00H | DPCON A7H 00H |
| SCON[1] 98H 00H | SBUF 99H 00H | I2CDAT 9AH 00H | I2CADD 9BH 55H | NOT USED | T3FD 9DH 00H | T3CON 9EH 00H | NOT USED |
| P1[1,2] 90H FFH | NOT USED | NOT USED | NOT USED | NOT USED | NOT USED | NOT USED | NOT USED |
| TCON[1] 88H 00H | TMOD 89H 00H | TL0 8AH 00H | TL1 8BH 00H | TH0 8CH 00H | TH1 8DH 00H | RESERVED | RESERVED |
| P0[1] 80H FFH | SP 81H 07H | DPL 82H 00H | DPH 83H 00H | DPP 84H 00H | RESERVED | RESERVED | PCON 87H 00H |

SFR MAP KEY:

THESE BITS ARE CONTAINED IN THIS BYTE.



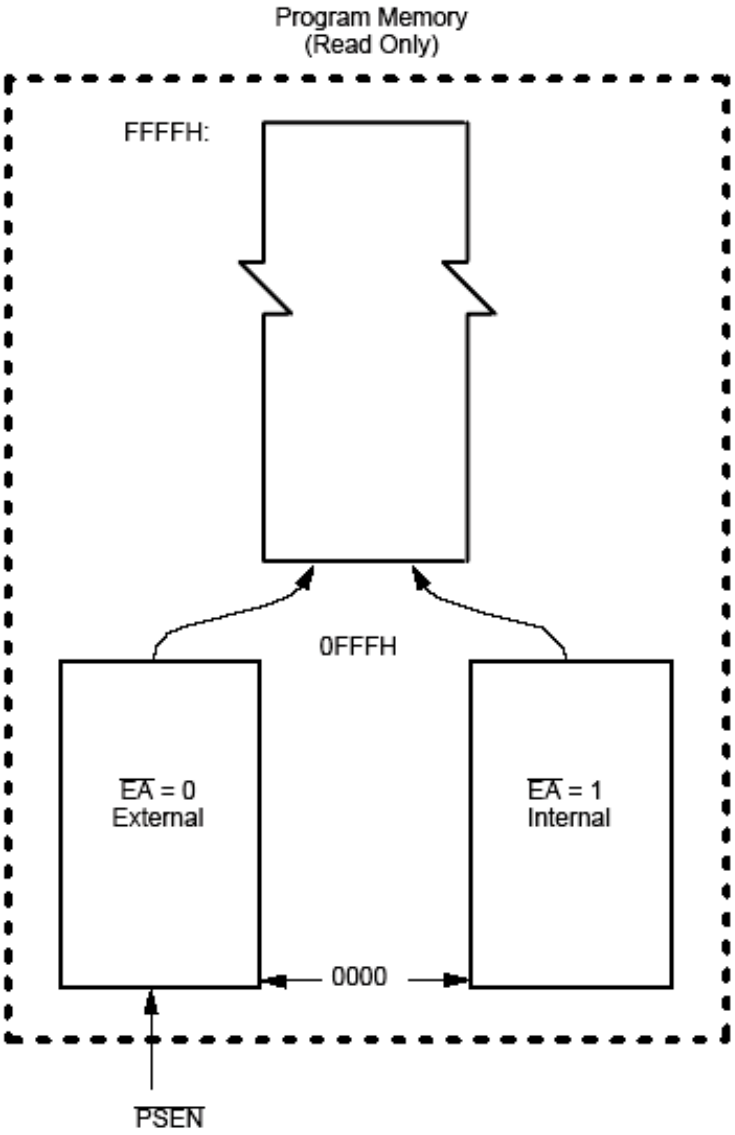

NOTES

[1]SFRs WHOSE ADDRESS ENDS IN 0H OR 8H ARE BIT ADDRESSABLE.
[2]THE PRIMARY FUNCTION OF PORT1 IS AS AN ANALOG INPUT PORT; THEREFORE, TO ENABLE THE DIGITAL SECONDARY FUNCTIONS ON THESE PORT PINS, WRITE A "0" TO THE CORRESPONDING PORT 1 SFR BIT.
[3]CALIBRATION COEFFICIENTS ARE PRECONFIGURED ON POWER-UP TO FACTORY CALIBRATED VALUES.

*Figure 6. Special Function Register Locations and Reset Values*

I registri indirizzabili per bit in questa area sono quelli il cui indirizzo termina per '0' o per '8'. In sostanza la prima metà degli '8' bit dell'indirizzo. L'istruzione per scrivere nel bit SETB bit o CLR bit. Si può compiere un movimento del tipo MOV C,bit o MOV bit,C. Dove C è il carry e bit è nome_registro.X con X=0,..,7.

# L'organizzazione della memoria dell'Intel 8051 5




Come in tutti i micro che hanno un certo ammontare di memoria programmi interna, c'è la possibilità di adottare diverse configurazioni.

All'accensione, o dopo una fase di reset, il micro deve partire sempre da una posizione di memoria prestabilita. Fa differenza se la memoria usata è solo esterna o meno.

Un pin viene dedicato alla definizione della memoria programmi utilizzata. Se il pin EA è '0' la memoria viene tutta considerata all'esterno. La cella 00, la posizione di partenza del programma all'accensione, è considerata localizzata nella memoria esterna.

Se EA=1 la memoria esterna è considerata quella che parte da un certo indirizzo in su. Nell'esempio sopra l'indirizzo della memoria esterna parte da FFFh+1. In questo caso alla partenza la prima cella di memoria considerata è la 000h, nella memoria interna.

Nell'8051 classico l'indirizzamento è a 16 bit, per cui la memoria si può estendere fino a 64 KB. Pur di aggiungere byte di indirizzamento e tempo di esecuzione si può cercare di indirizzare una quantità di memoria maggiore.

Ad esempio l'ADuC832 di Analog indirizza comodamente 16 MB di memoria esterna usando un byte addizionale. Inoltre la sua RAM interna ha 2 KB addizionali che possono essere indirizzati con l'artificio di aggiungere 3 bit di indirizzo al Data Pointer.

# Le porte di I/O dell'8051 1

Lo standard per il numero di porte dell'8051 è 4, sebbene ci siano casi in cui per minimizzare il numero di pin si hanno a disposizione meno porte, o addirittura di più.

Sebbene le 4 porte non abbiano la stessa configurazione circuitale, un solo registro è associato ad ogni porta nello standard 8051. Casi come con Philips si associano 3 byte: 2 byte servono per determinare la direzione scelta e la modalità di funzionamento per ogni pin della porta, ed uno per tenere il valore memorizzato nella porta, quando è assegnata come di uscita, o l'ultimo valore letto, quando configurata come ingresso.

Le 4 porte vengono chiamate Port0, Port1, Port2 e Port3.

Dal punto di vista HW le porte non sono identiche, come vedremo più avanti.

La memoria esterna viene indirizzata dalla Port0 (parte bassa indirizzo e dato) e Port2 (parte alta dell'indirizzo). Queste 2 porte non possono essere usate per altri scopi quando usate per gestire la memoria esterna.

La porta Port1 è una generica porta di I/O.

La porta Port3 può essere usata come generica porta di I/O, ma soddisfa anche altre funzionalità.

# Le porte di I/O dell'8051 2

La Port3 nell'8051 classico si occupa della gestione della comunicazione seriale, degli interrupt, dei pin del timer e dei segnali di lettura/scrittura dalle memorie esterne.

Le funzioni alternative a cui i pin della Port3 possono adempiere sono:

**Table XIX. Port 3, Alternate Pin Functions**

| Pin | Alternate Function |
|---|---|
| P3.0 | RxD (UART Input Pin)(or Serial Data I/O in Mode 0) |
| P3.1 | TxD (UART Output Pin)<br>(or Serial Clock Output in Mode 0) |
| P3.2 | $\overline{\text{INT0}}$ (External Interrupt 0) |
| P3.3 | $\overline{\text{INT1}}$ (External Interrupt 1)/PWM 1/MISO |
| P3.4 | T0 (Timer/Counter 0 External Input)<br>PWM External Clock/PWM 0 |
| P3.5 | T1 (Timer/Counter 1 External Input) |
| P3.6 | $\overline{\text{WR}}$ (External Data Memory Write Strobe) |
| P3.7 | $\overline{\text{RD}}$ (External Data Memory Read Strobe) |

Le complicazioni usate nell'8051 nella gestione delle porte non sono necessariamente riscontrati in altri micro. Questa complicazione è stata inizialmente adottata per risparmiare registri di configurazione per le porte. Infatti con un solo registro per porta si riesce a fare tutto.

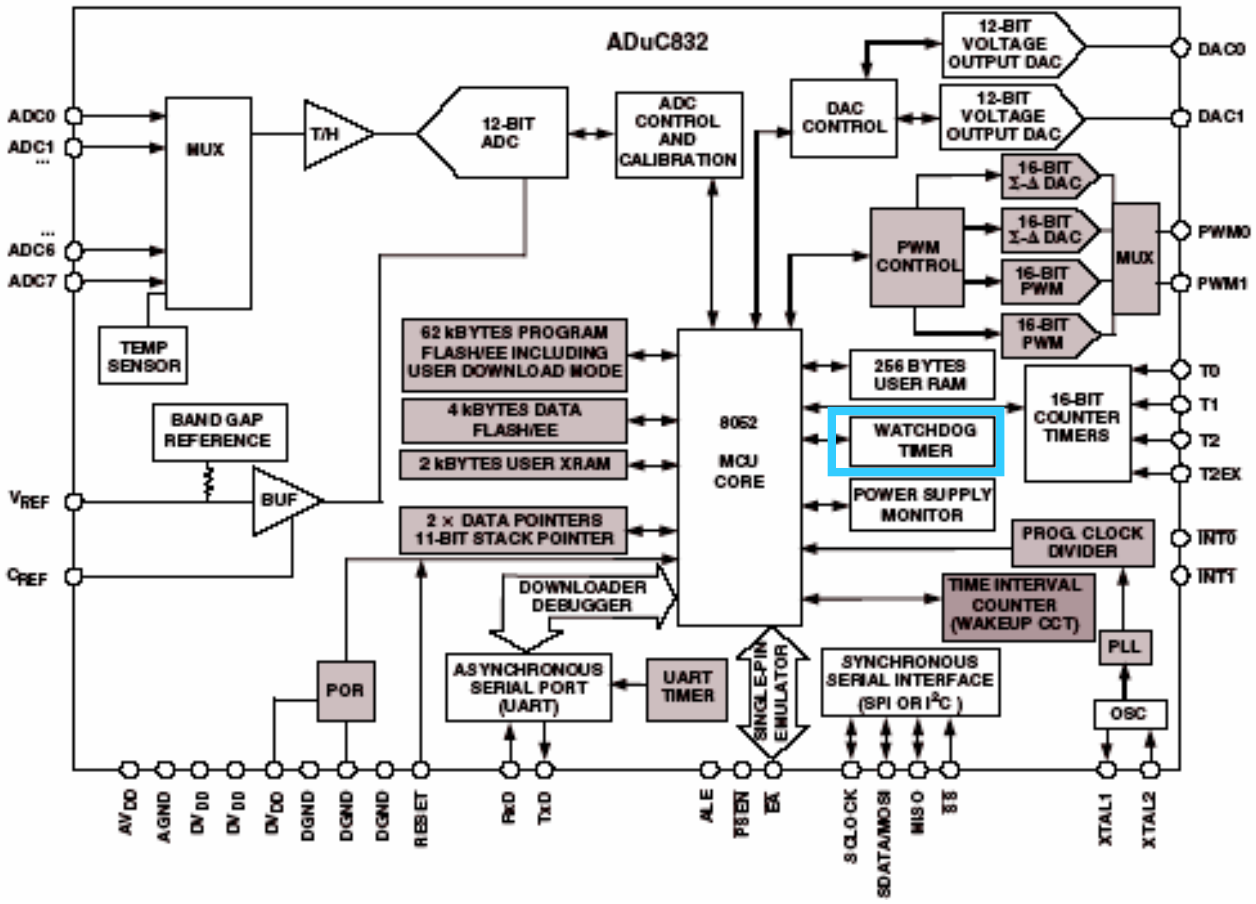
La possibilità di disporre di una maggiore quantità di RAM consente di potere spendere più registri per la gestione delle porte. Ad esempio i micro di Philips, pur essendo basati su 8051 dispongono di più registri di configurazione che consentono una maggiore flessibilità.

# Un 8051 complesso: Analog ADUC832

Si può osservare nello schema che il cuore del dispositivo è in effetti il micro basato su 8051. Vi sono molte periferiche aventi funzionalità analogiche: ADC e DAC, nonché comparatori.

Ovviamente ognuna di queste periferiche va impostata. Per ogni impostazione occorre dedicare qualche bit di qualche registro.

In questo micro vi è presente una memoria addizionale RAM interna. Secondo la filosofia dell'8051 questa memoria è vista come una memoria esterna all'interno del micro.




C'è un ulteriore dispositivo importante: il WATCHDOG-TIMER. E' un dispositivo digitale che sovrintende al funzionamento del micro. Questo dispositivo si aspetta che ad intervalli regolari il micro cambi lo stato di una linea ad esso connessa. Se questa azione non venisse svolta, automaticamente verrebbe dato un impulso di reset. Ovvero la mancata notifica dell'impulso da parte del micro viene interpretata come un sintomo di malfunzionamento che si intende curare con una ri-partenza del sistema.

All'accensione il WATCHDOG agisce mantenendo bassa la linea di reset per il tempo necessario al raggiungimento del regime del sistema.

# Gli Interrupt nell'8051 1

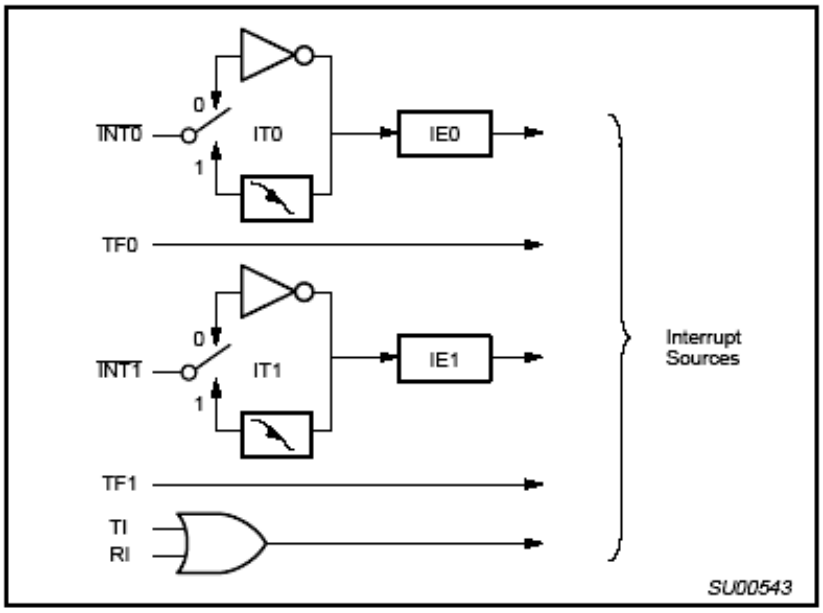



Figure 17. 80C51 Interrupt Sources

L'8051 ha almeno 5 interrupt, in aggiunta al reset pin. 2 di queste interrupt sono SW e derivano dai 2 timer, TF0 e TF1. Delle altre 3, 2 provengono da 2 pin dedicati Not_INT0 e Not_INT1. La terza dalla porta di ingresso seriale, un OR tra 2 linee.

Esiste un registro, TCON, dove si può impostare il tipo di ingresso che determina la risposta all'interrupt. Vale a dire se il segnale di eccitazione deve essere riconosciuto sul livello o sulla transizione.

Ogni segnale di interrupt può essere abilitato o meno al servizio, mascherato o s-smascherato. Basta impostare il bit appropriato nel registro IE (Interrupt Enable).

MSB LSB

| EA | X | X | ES | ET1 | EX1 | ET0 | EX0 |
|---|---|---|---|---|---|---|---|

| BIT | SYMBOL | FUNCTION |
|---|---|---|
| IE.7 | EA | Disables all interrupts. If EA=0, no interrupt will be acknowledged. If EA=1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit. |
| IE.6 | — | Reserved. |
| IE.5 | — | Reserved. |
| IE.4 | ES | Enables or disables the Serial Port interrupt. If ES=0, the Serial Port interrupt is disabled. |
| IE.3 | ET1 | Enables or disables the Timer 1 Overflow interrupt. If ET1=0, the Timer 1 interrupt is disabled. |
| IE.2 | EX1 | Enables or disables External Interrupt 1. If EX1=0, External interrupt 1 is disabled. |
| IE.1 | ET0 | Enables or disables the Timer 0 Overflow interrupt. If ET0=0, the Timer 0 interrupt is disabled. |
| IE.0 | EX0 | Enables or disables External interrupt 0. If EX0=0, External interrupt 0 is disabled. |

SU00544

Figure 18. Interrupt Enable Register (IE)

# Gli Interrupt nell'8051 2

La priorità con cui i segnali di interrupt vengono serviti è impostabile nel registro IP(Interrupt Priority). Esiste poi, elencata, tutta una serie di regole per gestire le equi-priorità.

| MSB | | | | | | | LSB |
|---|---|---|---|---|---|---|---|
| X | X | X | PS | PT1 | PX1 | PT0 | PX0 |

| BIT | SYMBOL | FUNCTION |
|---|---|---|
| IP.7 | — | Reserved. |
| IP.6 | — | Reserved. |
| IP.5 | — | Reserved. |
| IP.4 | PS | Defines the Serial Port interrupt priority level. PS=1 programs it to the higher priority level. |
| IP.3 | PT1 | Defines the Timer 1 interrupt priority level. PT1=1 programs it to the higher priority level. |
| IP.2 | PX1 | Defines the External Interrupt 1 priority level. PX1=1 programs it to the higher priority level. |
| IP.1 | PT0 | Enables or disables the Timer 0 interrupt priority level. PT0=1 programs it to the higher priority level. |
| IP.0 | PX0 | Defines the External Interrupt 0 priority level. PX0=1 programs it to the higher priority level. |

SU00545

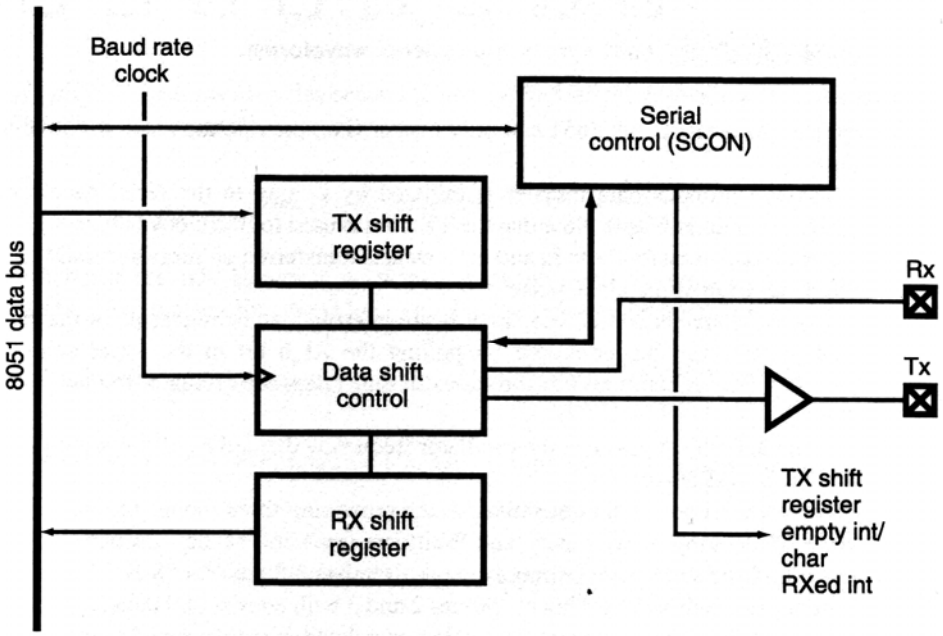
Figure 19. Interrupt Priority Register (IP)

| Source | Vector Address |
|---|---|
| IE0 | 0003H |
| TF0 | 000BH |
| IE1 | 0013H |
| TF1 | 001BH |
| RI+TI | 0023H |

Una volta che l'interruzione è stata riconosciuta viene lanciata la subroutine il cui indirizzo di partenza è definito a priori.

Per ogni chiamata si hanno a disposizione 8 celle nella memoria programmi. Se il servizio richiedesse una maggiore quantità di memoria basterebbe quindi inserire un salto in-condizionato, in una delle 8 celle a disposizione, ad una zona opportuna della memoria programmi.

# Il funzionamento della seriale (UART) 1




8051 serial Rx/Tx block diagram.

La trasmissione seriale è un tipico esempio di protocollo che può essere gestito su interrupt.

Nell'8051 è organizzata con 2 registri: il registro di trasmissione e quello di ricezione.

**TABLE 4-9 DESCRIPTION OF THE SCON REGISTER**

| BIT | DESCRIPTION |
|---|---|
| 0 | RI—Set by hardware when a character has been received |
| 1 | TI—Set by hardware when the stop bit is being transmitted |
| 2 | RB8—Ninth data bit received in modes 2 and 3 |
| 3 | TB8—Ninth data bit sent by software in modes 2 and 3 |
| 4 | REN—Set to enable serial data reception |
| 5 | SM2—Enable multi-processor communications mode |
| 6–7 | SM1/SM0—Serial port mode bits<br>00—Mode 0, synchronous serial communications<br>01—Mode 1, 8-bit UART with timer data rate<br>10—Mode 2, 9-bit UART with set data rate<br>11—Mode 3, 9-bit UART with timer data rate |

La prima cosa che occorre fare è abilitare la ricezione nello SCON register (bit REN). Si devono impostare anche i bit SM0/SM1 per determinare la modalità di trasmissione: sincrona, asincrona, ecc.

L'ultima cosa da abilitare è la velocità di trasmissione, determinata attraverso il bit 7 dello SMOD register.

A questo punto la comunicazione può essere gestita seguendo 2 modalità: mediante polling o interrupt.

Trasmissione e ricezione in polling:

- Il dato da trasmettere viene posto nel registro TX di trasmissione SBUF (099h). Automaticamente viene trasmesso. Alla fine della trasmissione il bit TI dello SCON è settato. Una volta accertato, TI occorre resettarlo via SW per adempiere alla successiva trasmissione.
- Il dato ricevuto è posto nello SBUF, dopo che la ricezione è stata abilitata. Il pin RI è allora alzato dal micro. Una volta accertato, RI occorre resettarlo via SW per adempiere alla successiva ricezione.

# Il funzionamento della seriale (UART) 2

**TABLE 4-5 DESCRIPTION OF THE IE REGISTER**

| BIT | DESCRIPTION |
|---|---|
| 0 | EX0—_INT0 pin interrupt enable |
| 1 | ET0—Timer0 overflow interrupt enable |
| 2 | EX1—INT1 pin interrupt enable |
| 3 | ET1—Timer1 overflow interrupt enable |
| 4 | ES—Serial port interrupt enable |
| 5–6 | Reserved for additional interrupt hardware |
| 7 | IE—Global interrupt enable |

Per sfruttare l'interrupt occorre abilitare la seriale all'interruzione nel registro IE. A questo punto tanto la fine ricezione che la fine trasmissione generano interrupt, oltre che a dovere essere impostate e comportarsi come nel caso del polling.

Va osservato che: il registro di ricezione coincide con quello di trasmissione. Occorre fare attenzione a che la ricezione sia finita e si sia spostato il dato prima di scriverci dentro.

La scrittura di un dato nello SBUF mentre si sta trasmettendo genera confusione, è un'azione da evitare.

Anche quando si genera l'interrupt i 2 bit RI e TI devono essere resettati via SW, non vengono fatti via HW come tutti gli altri tipi di interrupt.

La trasmissione è sempre abilitata, mentre la ricezione lo è solo quando REN e RI sono bassi.

**IMPORTANTE**:

Nel protocollo seriale solo 2 interlocutori possono essere in comunicazione. La comunicazione seriale è solo punto-punto.

# La comunicazione I²C 1

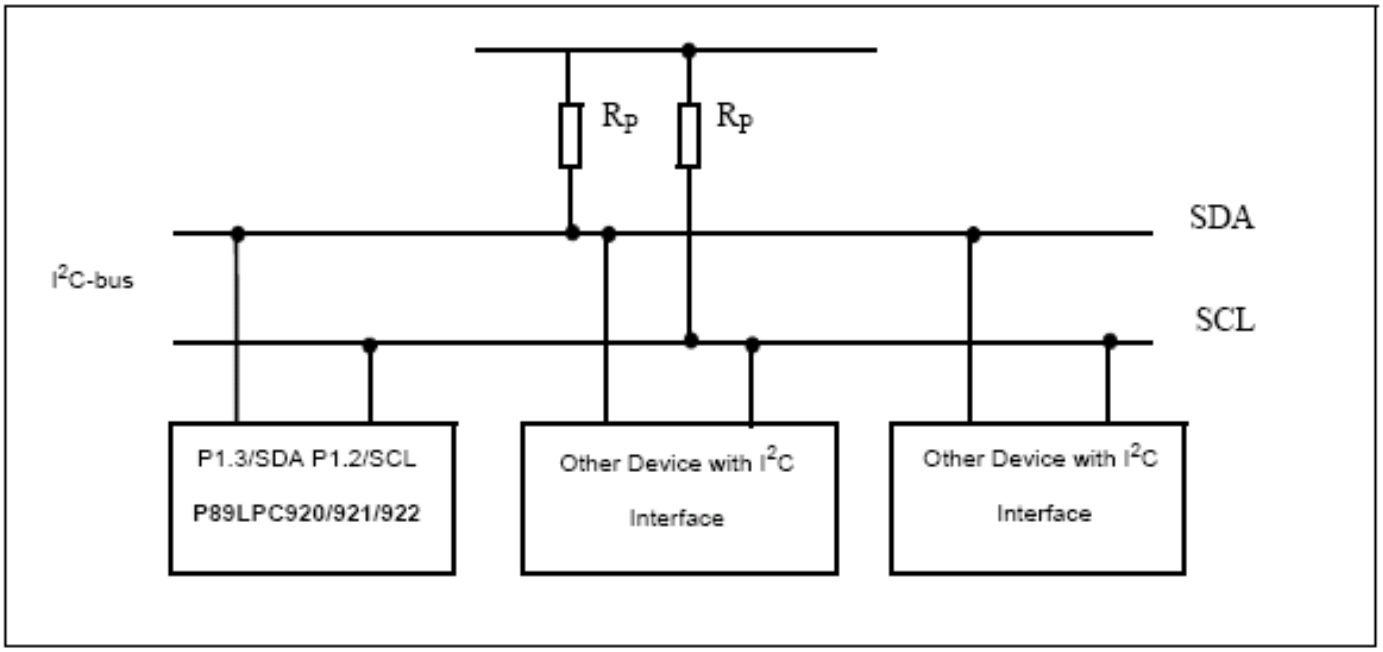
Il protocollo I²C che sta per Inter-Inter Computer Communication. Fu sviluppato da Philips negli anni '70.

E' un protocollo che necessita di sole 2 linee per realizzarsi. Molte unità possono accedere al bus. Ogni unità avrà il proprio indirizzo che la individuerà.

Sebbene abbia solo 2 fili consente una comunicazione di tipo Master-Slave e Multi-master.

Nel primo caso abbiamo un'unità che è responsabile di interrogare le unità dopo averle indirizzate.

Nel secondo caso ogni unità può prendere il sopravvento del bus, quando necessario, e comunicare con un'altra unità che selezionerà. Nel caso più unità vogliano diventare Master allo stesso tempo un processo di arbitrariato darà la priorità a quella più 'importante'.




Le 2 linee necessitano di 2 resistenze dette di pull-up. Le linee devono essere open-drain (vedremo cosa significherà).

Una delle 2 linee è detta SCL (Serial Clock), ovvero il clock. L'altra linea è detta SDA (Serial Data), ovvero il dato.

# La comunicazione I²C 2

La velocità di comunicazione di questo tipo di protocollo può arrivare ad 1 M Hz. La particolarità è che la velocità di trasmissione non è fissata. La connessione open-drain consente di fare si che una periferica lenta, quando in comunicazione, possa 'rallentare' il clock per soddisfare i propri requisiti.

Di principio sarebbe possibile costruire il protocollo I²C. Tuttavia questo tipo di protocollo è spesso già implementato nel micro che mette a disposizione 2 pin adibiti allo scopo.

I 2 pin sono connessi ad una logica interna al micro che automaticamente è in grado di fare fronte a tutte le situazioni che si possono presentare.

E' possibile che si possano verificare fino a 26 situazioni diverse.

Una volta stabilita una comunicazione lo **Status Register** può essere interrogato e consente di verificare cosa sta avvenendo.

**I2STAT**

Address: D9h

Not bit addressable

Reset Source(s): Any reset

Reset Value: 11111000B

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| STA.4 | STA.3 | STA.2 | STA.1 | STA.0 | 0 | 0 | 0 |

| BIT | SYMBOL | FUNCTION |
|---|---|---|
| I2STAT7, 3 | STA.4, 0 | I²C the status code. |
| I2STAT2, 0 | - | These three bits are not used and always set to 0. |

**Figure 5: I²C Status register**

Accanto allo Status Register il protocollo comprende anche:

- il **Data Register**, dove viene posto il dato da spedire o il dato ricevuto,
- **l'Address Register**, che contiene il proprio indirizzo se si è slave,
- il **Control Register,** che consente di impostare le impostazioni in uso.

# La comunicazione I²C 3

Come si può verificare, dietro ai 2 pin di comunicazione è necessaria la presenza di una struttura logica complessa in grado di gestire il protocollo.

Nello sviluppo il programma dovrà prevedere tutti i casi che si possono fronteggiare nella comunicazione, onde minimizzare ogni possibile condizione di errore.

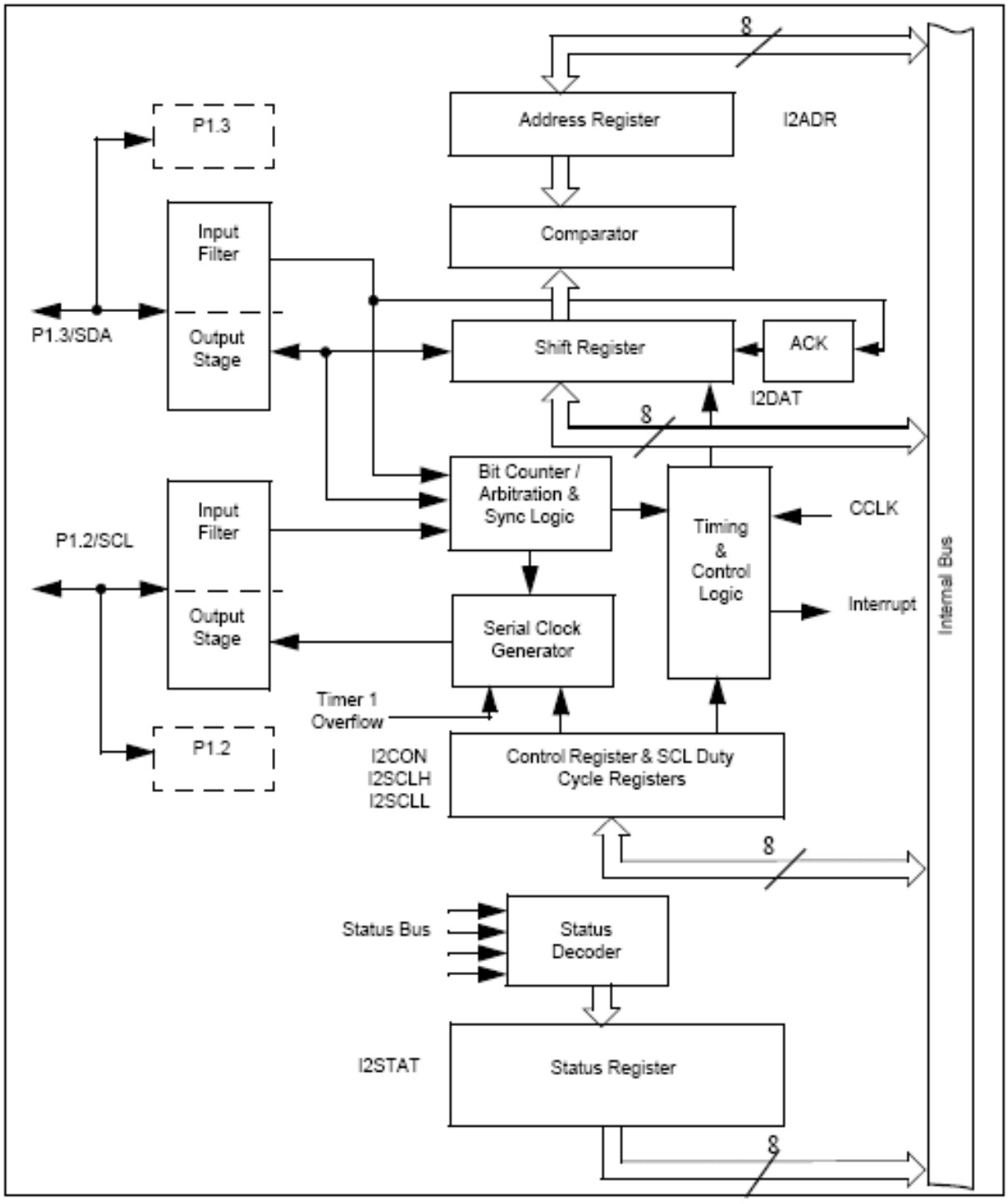



Figure 13: I²C-bus serial interface block diagram

# La comunicazione I²C 4

La sequenza di comunicazione deve seguire uno standard ben preciso:

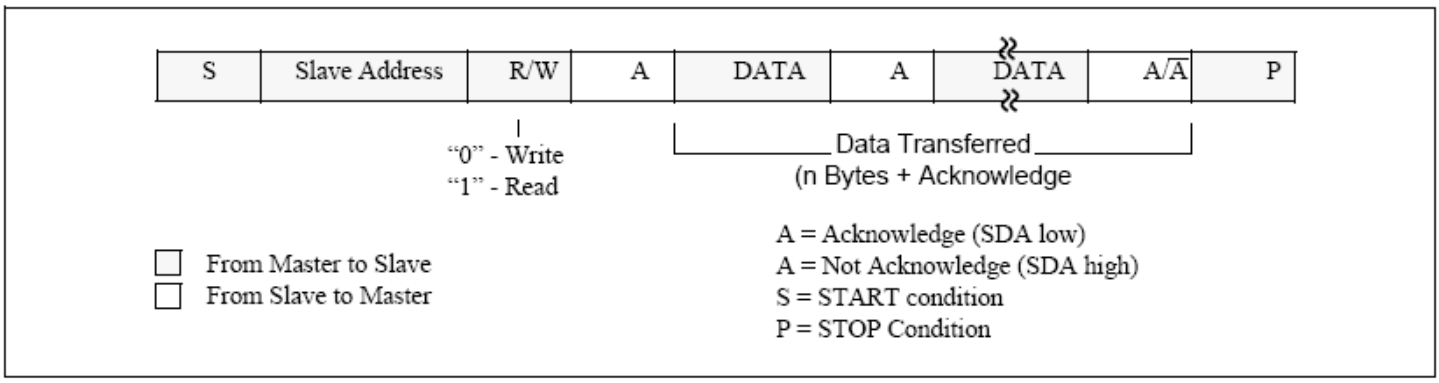



**Figure 11-7: Format in the Master Transmitter Mode**

La comunicazione parte con lo start bit, S. Viene poi emesso l'indirizzo di 7 bit della periferica a cui si vuole parlare. Il successivo bit comunica se si intende spedire un dato o ricevere un dato, R/W.

A questo punto si attende un segnale di riconoscimento, A, dalla periferica.

Ricevuto il riconoscimento si inizia a mandare o ricevere uno o più dati.

Alla fine della comunicazione viene emesso uno stop bit, P.

# Il protocollo SPI (Serial Periferal Interface) 1

SPI è un protocollo molto semplice ed efficiente che consente di trasferire dati a velocità anche di qualche MHz o più.

E' un protocollo di comunicazione dove è presente un master che può interrogare diversi slave. Ogni slave non è caratterizzato da un indirizzo, ma è selezionato dal master mediante una linea dedicata.

il numero di linee necessarie è di 3 più una linea per ogni slave.

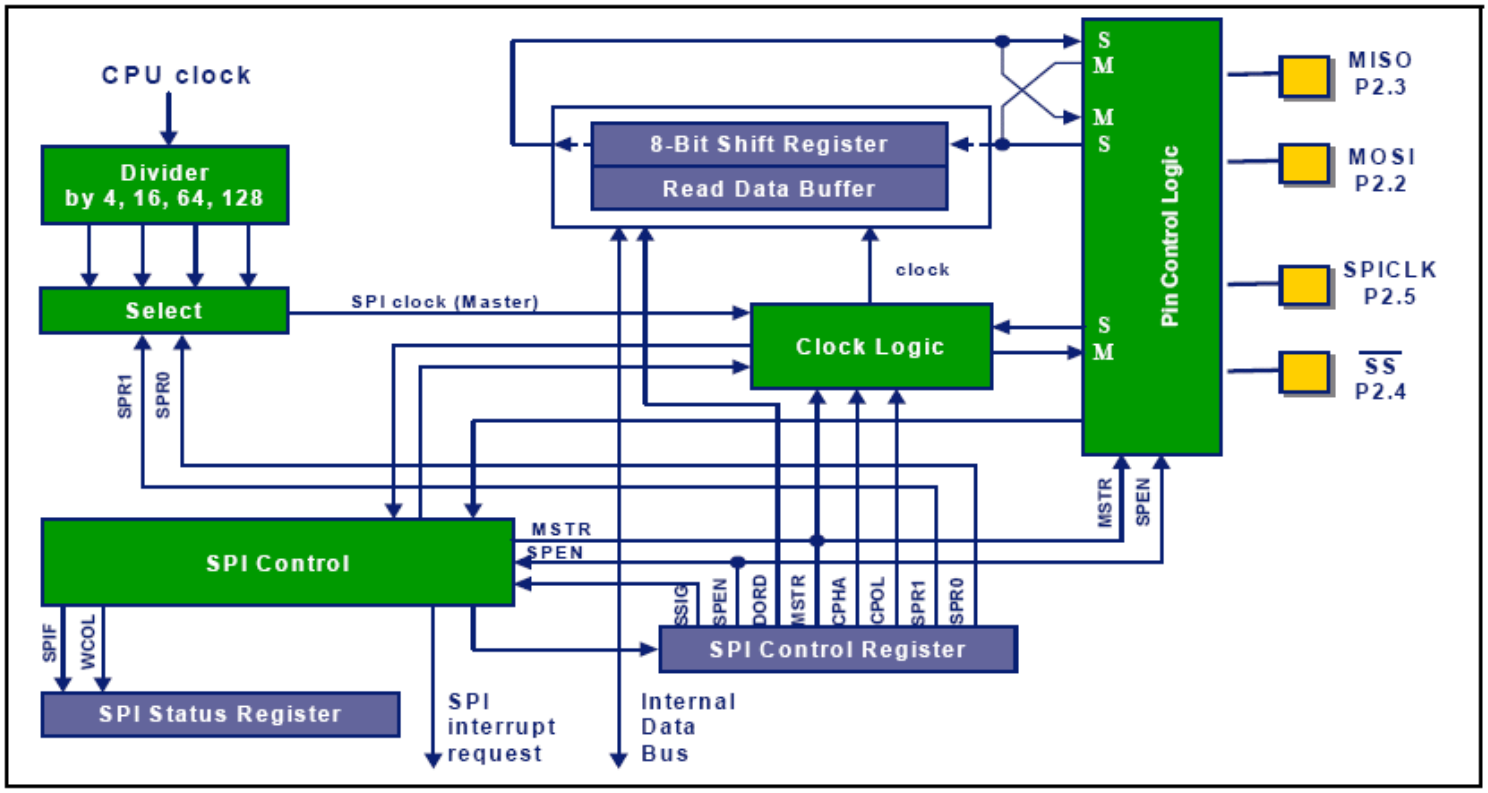



Figure 12-1: SPI block diagram

Anche qui c'e' una logica complessa che gestisce i pin di comunicazione. Le linee necessarie sono:

- **SPICLK**: il clock di comunicazione. Il protocollo è sincrono;
- **MISO**: sta per Master In Slave Out è la linea di ingresso dei dati al master.
- **MOSI**: sta per Master Out Slave In è la linea di dati uscenti dal master.
- **SS**: è la (le) linea(e) di selezione dello slave.

# Il protocollo SPI (Serial Periferal Interface) 2

Anche qui abbiamo dei registri che sono adibiti alla gestione della comunicazione:

- Il **Control Register (SPCTL)** consente di impostare le modalità di funzionamento;
- Lo **Status Register (SPSTAT)** segnala lo stato della trasmissione;
- **Il Data Register (SPDAT)** è il registro dove è posto il dato spedito e quello letto alla fine della trasmissione.

Il collegamento al singolo slave viene fatto così:

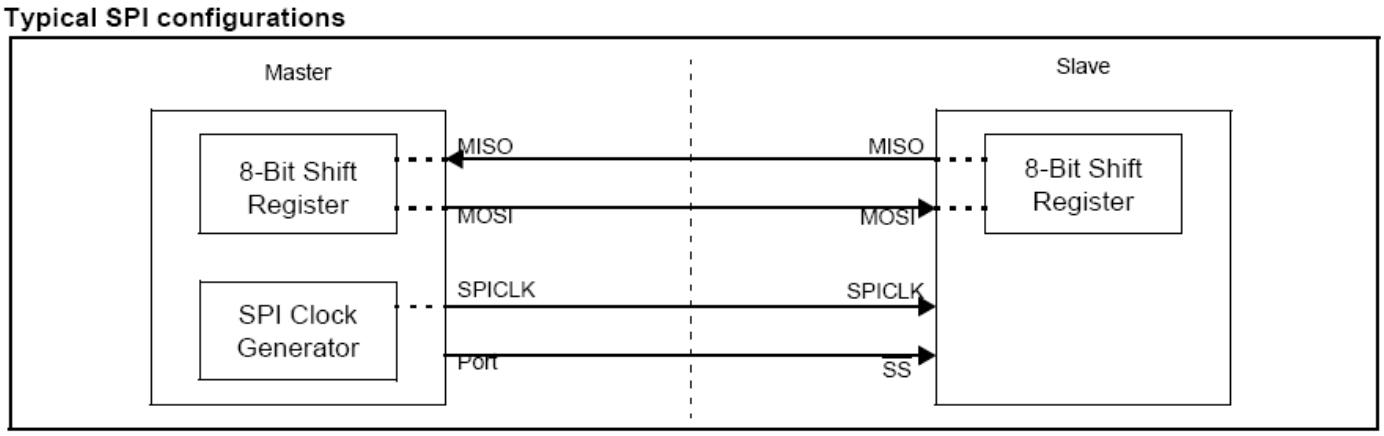



**Figure 63: SPI single master single slave configuration**

Nel caso un singolo slave sia connesso il pin SS è ridondante.

# Il protocollo SPI (Serial Periferal Interface) 3

Nel caso di comunicazione a più slave occorre la presenza di più bit per l'abilitazione della periferica con cui colloquiare:

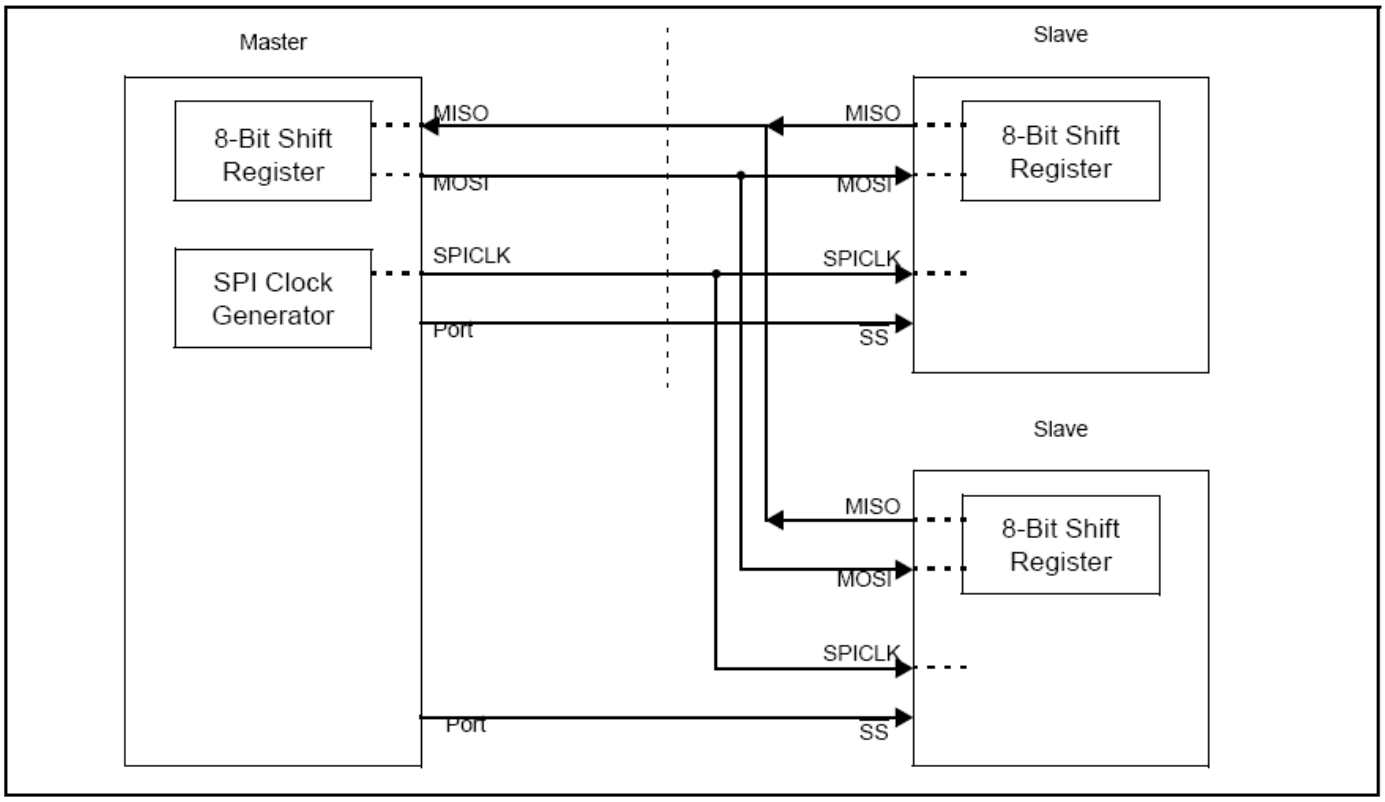



**Figure 65: SPI single master multiple slaves configuration**

# Il protocollo SPI (Serial Periferal Interface) 4

Nella comunicazione ad ogni trasmissione corrisponde una ricezione. E' come se il registro dati in uscita da una parte scorra nell'altra e viceversa.

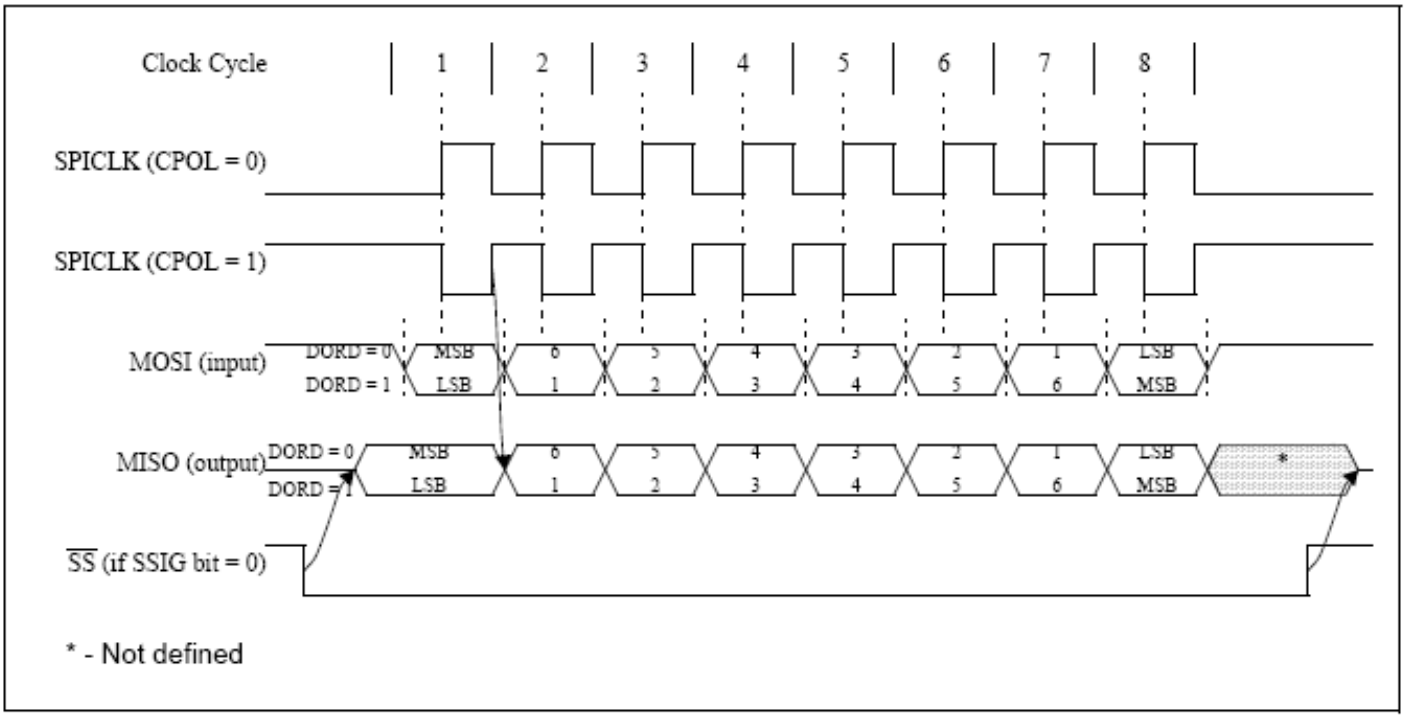



Figure 66: SPI slave transfer format with CPHA = 0

Si può osservare come la comunicazione sia arbitrata a seconda della impostazione di alcuni bit, CPHA, SSIG, etc.

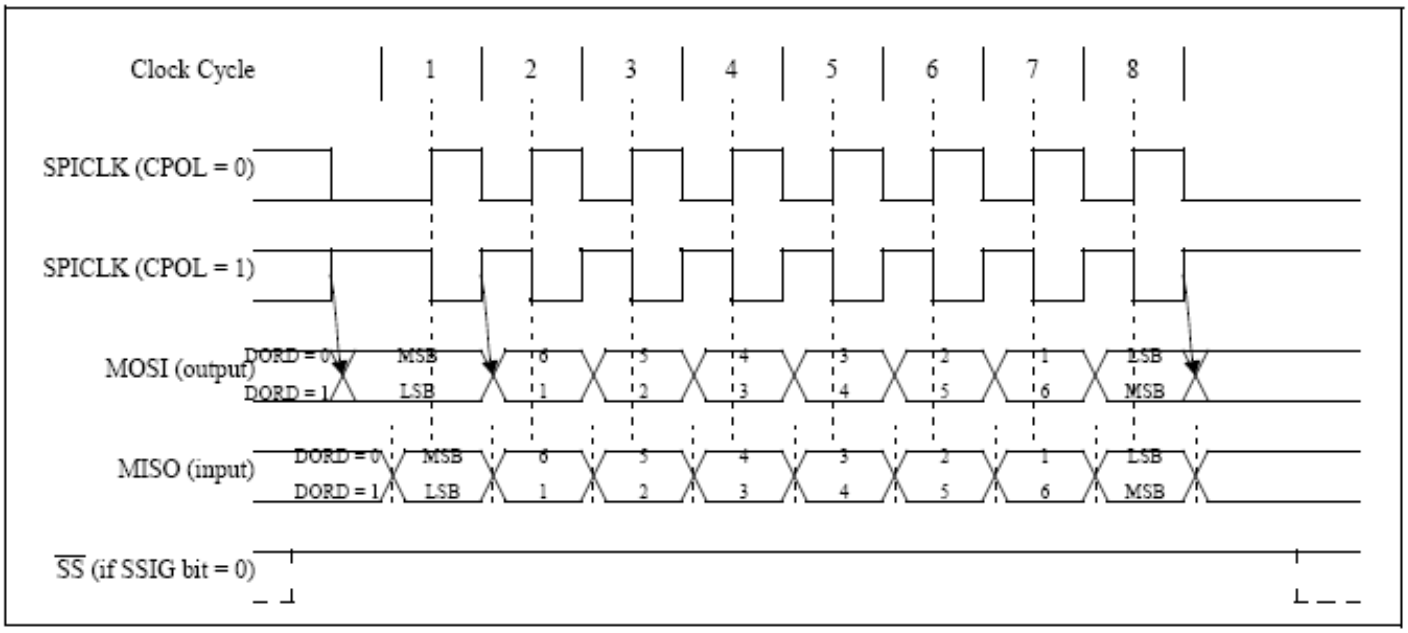



Figure 68: SPI master transfer format with CPHA = 0

# Altri protocolli di comunicazione

I protocolli **SPI** e **$I^2C$** sono tipicamente impiegati nella comunicazione tra dispositivi. Non sono gli unici disponibili, ma sono più comuni e semplici da utilizzare.

Esistono anche altre protocolli. Ad esempio il protocollo **CAN (Controller Area Network)** è sfruttato tipicamente nell'automotive.

Il **μ-wire** è un bus a singolo filo introdotto da Maxim-Dallas.

Accanto ai protocolli per comunicazione tra dispositivi ci sono anche quelli per comunicazione con il mondo 'reale'.

Il protocollo seriale standard, **UART**, è il più semplice ed utilizzato.

Attualmente molti micro-controllori di nuova generazione sono già in grado di gestire la comunicazione con **USB** e **LAN** o **TCP/IP**.

# L'ambiente di sviluppo di un micro 1

Lo sviluppo di un progetto per un micro comprende diverse fasi successive che vanno scelte con un certo criterio e cura onde evitare di trovarsi di fronte a risorse inadeguate quando si è in una fase avanzata dello sviluppo.

Il primo passo è la selezione del tipo di micro. Questo è senz'altro il punto più critico. La scelta deve essere in grado di prevedere il dispositivo che meglio riesce ad interpretare le risorse HD necessarie.

A questo punto va osservato che una volta scelto il dispositivo occorre considerare che sistema di sviluppo esiste: se il costo è accessibile, se ha caratteristiche note, se consente una potenza di sviluppo adeguata, ecc.

Attualmente quasi sempre si verifica che qualsiasi micro si scelga esiste una scheda di sviluppo che contempla un SW.

Per potere risparmiare tempo occorre sempre munirsi dei seguenti documenti:

1. Datasheet del micro che illustra le caratteristiche HD disponibili ed utilizzabili;
2. HD manual: è il documento che descrive in dettaglio come usare le risorse del micro con un programma di sviluppo: elenca i registri di configurazione, come funzionano le periferiche, ecc.
   Spesso, se il dispositivo non è particolarmente sofisticato, l'HD manual ed il datasheet coincidono.
3. Il manuale del SW. In genere il SW è capace di gestire una famiglia di micro. Occorre riferirsi al suo manuale per selezionare le opzioni necessarie. Il manuale del SW è quasi sempre disponibile in linea.

# L'ambiente di sviluppo di un micro 1

Il SW associato al micro, o acquistato, capace di gestire il micro si chiama Integrated Development Environment, o IDE. L'IDE consente di svolgere tutti i passi che servono alla realizzazione del progetto. Attualmente tutti gli IDE consento di sviluppare il proprio progetto sia usando come linguaggio di programmazione l'Assembler che il C, o entrambi contemporaneamente.

Alla fine della stesura del programma il progetto va interpretato. In questa fase l'IDE ci segnalerà tutti gli errori e le incoerenze usate nella stesura. Il numero degli errori non è in genere proporzionale alla propria incapacità ma alla complessità del progetto.

Il progetto potrebbe essere composto di più di una file: più C o Assembly file, library file, include o header file, ecc. I library ed include file possono contenere definizioni di variabili pertinenti al micro, pezzi di programmi di uso generale ecc.

L'IDE si preoccuperà di unire o "linkare" tutti questo file per generare un unico file che dovrà essere sottoposto nella memoria programmi del micro.

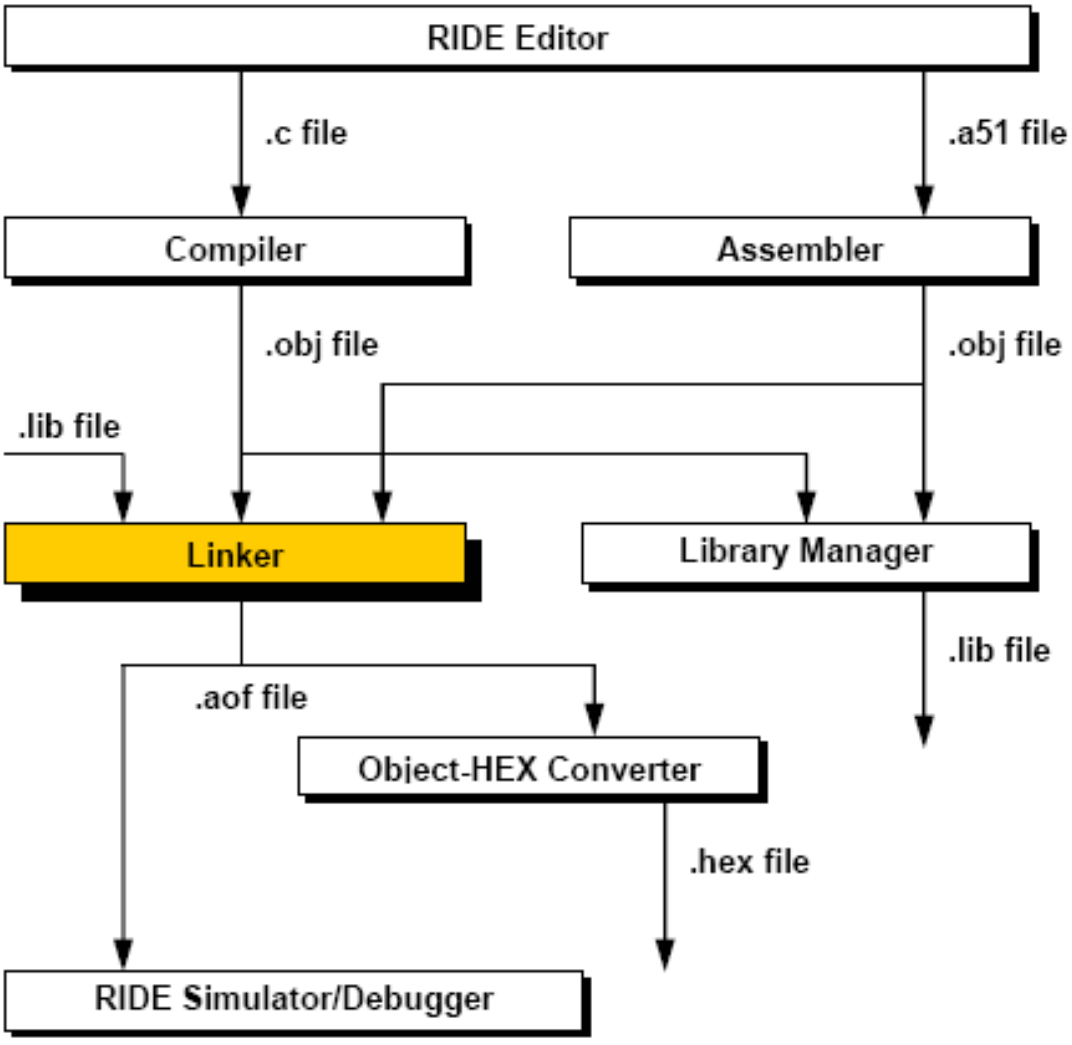
Tutti gli attuali IDE operano in ambiente Windows, che consente la gestione per finestre di tutte le fasi del processo di sviluppo.

Occorre infine considerare che spesso IDE prodotti da differenti società sono in grado di gestire il micro acquistato, specialmente se si tratta dell'8051, micro molto comune. E' perciò possibile compiere una scelta particolareggiata.

# L'ambiente di sviluppo di un micro 2

L'IDE (RIDE nell'esempio sotto) compila i file scritti in C ed assembla i file scritti in Assembly per creare dei file di tipo .obj, tradotti in un linguaggio comune di basso livello.

Se il progetto comprende più file la fase di linking consente l'unione di tutti i file per creare un unico file .aof. Se il progetto si ritenesse di uso generale sarebbe possibile addirittura creare un file .lib, che potrebbe essere possibile usare in un'altra applicazione.



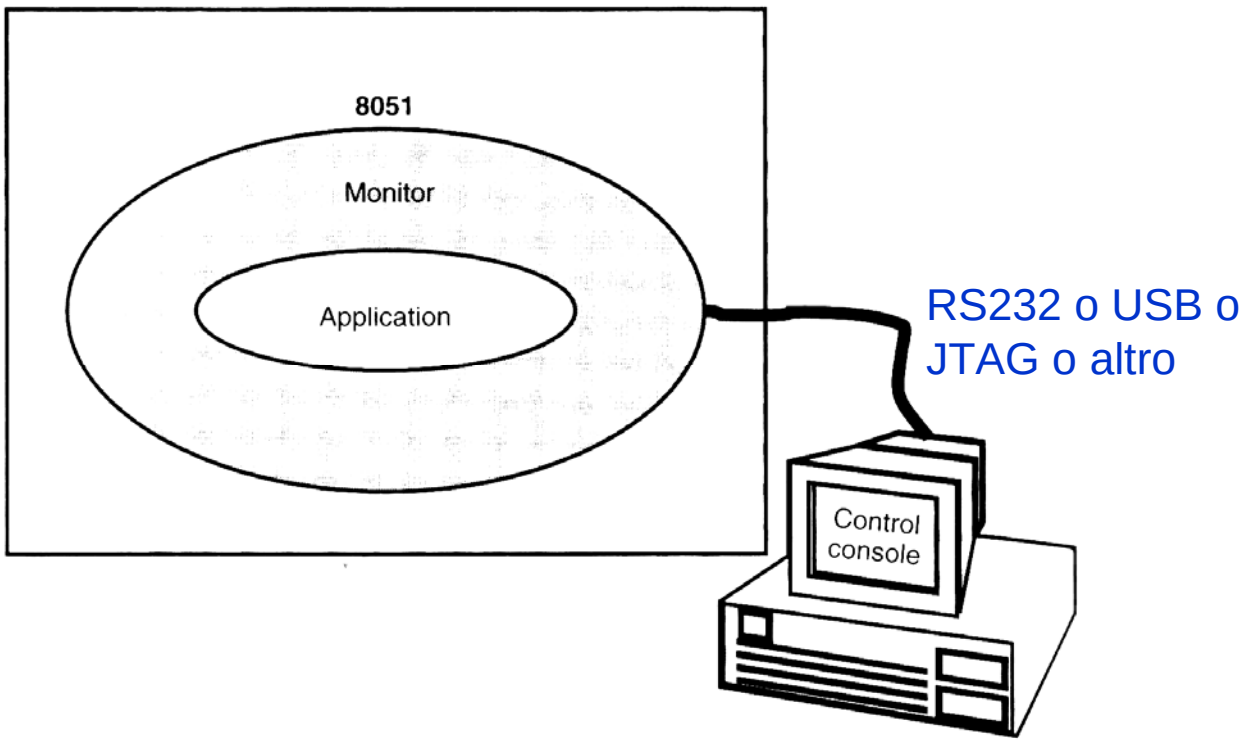

Dopo la fase di linking occorre verificare la correttezza del programma prima di caricarlo nel micro in forma definitiva. Questa fase si distingue in 2 parti: simulazione ed emulazione.

La simulazione non necessita la presenza di schede. L'IDE simula il comportamento del micro. Occorre dare un insieme di stimoli, o eccitazioni, che simulano il mondo esterno connesso alle porte del micro. L'IDE si preoccuperà di generare le risposte agli stimoli che potranno essere confrontate con quanto aspettato.

# L'ambiente di sviluppo di un micro 3

La fase di simulazione non è in genere sufficiente per potere verificare la correttezza del programma. L'evoluzione temporale del sistema è simulato. Però potrebbe verificarsi una qualsiasi causa improvvisa esterna che potrebbe essere non stata contemplata. Conviene allora emulare il dispositivo.

Sostanzialmente la condizione che si vorrebbe ricreare è il circuito finale dove al posto del micro sostituiamo le uscite di una schedina che simula in HD il comportamento del micro stesso. Questa schedina sarà connessa ad un PC che andrà a verificare il comportamento del programma. La schedina di emulazione sarà tanto più fedele alla realtà quanto più sarà in grado di mantenere le stesse condizioni di velocità del micro stesso, condizione spesso di difficile attuazione.




Tanto più la schedina sarà sofisticata, tanto più ci consentirà di svolgere il compito in un tempo breve. Ad esempio la possibilità di introdurre break-point, la possibilità di verificare il contenuto della memoria del micro mentre il programma è in esecuzione, ecc.

Cosa importante: i pin del micro utilizzati dal PC nella scheda di emulazione non sono utilizzabili. Per cui occorre fare attenzione al tipo di emulatore in rapporto alle risorse che si intendo usare del micro. Esempio: se la scheda comunica con il PC con la seriale non si potrà emulare la seriale del micro.

# L'Assembler 1

La conoscenza completa dell'assembler dipende molto anche dalla sintassi usata dall'IDE scelto.

Esistono tuttavia delle nozioni generali che vanno imparate onde poterle poi identificare.

I controlli od istruzioni si dividono in 3 grandi categorie: i controlli, le direttive e le istruzioni assembler vere e proprie.

Le istruzioni di controllo servono per comunicare al compilatore dove andare a prendersi le risorse e come impostare i prodotti del lavoro svolto:

| | | | |
|---|---|---|---|
| $DATE(date) | Places date in page header | $EJECT | Places a form feed in listing |
| $INCLUDE(file) | Inserts file in source program | $LIST | Allows listing to be output |
| $NOLIST | Stops outputting the listing | $MOD51 | Uses 8051 predefined symbols |
| $MOD52 | Uses 8052 predefined symbols | $MOD44 | Uses 8044 predefined symbols |
| $NOMOD | No predefined symbols used | $OBJECT(file) | Places object output in file |
| $NOOBJECT | No object file is generated | $PAGING | Break output listing into pages |
| $NOPAGING | Print listing w/o page breaks | $PAGELENGTH(n) | No. of lines on a listing page |
| $PAGEWIDTH(n) | No. of columns on a listing page | $PRINT(file) | Places listing output in file |
| $NOPRINT | Listing will not be output | $SYMBOLS | Append symbol table to listing |
| $NOSYMBOLS | Symbol table will not be output | $TITLE(string) | Places string in page header |

Table 2-1: Summary of Cross Assembler Controls

Es.:

```
$TITLE(8051 Program Ver. 1.0)
$LIST
$PAGEWIDTH(132)
```

Queste istruzioni sono in genere precedute dal simbolo $.

Va osservato che a seconda dell'IDE utilizzato e del dispositivo la sintassi potrebbe cambiare sensibilmente, pur restando inalterate le funzionalità di base.

# L'Assembler 2

Le istruzioni direttive sono molo importanti. Sono comandi che vengono soddisfatti dal compilatore che sulla base della loro impostazione organizza la memoria dati e programmi del micro:

| | | | |
|---|---|---|---|
| EQU | Define symbol | DATA | Define internal memory symbol |
| IDATA | Defin indirectly addressed internal memory symbol | XDATA | Define external memory symbol |
| BIT | Define internal bit memory symbol | CODE | Define program memory symbol |
| DS | Reserve bytes of data memory | DBIT | Reserve bits of bit memory |
| DB | Store byte values in program memory | DW | Store word values in program memory |
| ORG | Set segment location counter | END | End of assembly language source file |
| CSEG | Select program memory space | DSEG | Select internal memory data space |
| XSEG | Select external memory data space | ISEG | Select indirectly addressed internal memory space |
| BSEG | Select bit addressable memory space | IF | Begin conditional assembly block |
| ELSE | Alternative conditional assembly block | ENDIF | End conditional assembly block |
| USING | Select register bank | | |

Table 2-2: Summary of Cross Assembler Directives

Si possono dichiarare variabili od associare dei simboli sia a delle celle di memoria programma che dati.

Mediante i comandi DSEG e CSEG si definiscono aree di memoria dove porre dati o parti di programmi. Le parti di programmi possono essere sia rilocabili (il compilatore decide dove porli all'atto della compilazione)

***Syntax:*** **RSEG** segment_name

```
Example:      ;

mycodseg      SEGMENT  CODE   INBLOCK
              ; this relocatable code segment has a size less
              ; than 2048 bytes
mydat1seg     SEGMENT  DATA
              ; mydat1seg DATA segment declaration
mydat2seg     SEGMENT  DATA
              ; SEGMENT statement allows several segment
              ; declaration even with the same type
RSEG      mycodseg
              ; mycodseg CODE segment selection
  MOV     TMOD,#20h
  MOV     TCON,#44h
              ; instructions within the mycodseg CODE segment
RSEG      mydat1seg
              ; mydat1seg DATA segment selection
  COUNTER:    DB   1
              ; memory allocation
RSEG      mycodseg
              ; mycodseg CODE segment may be reselected
              ; the address pointer in mycodseg is automatically
              ; increased depending on the instruction's length
  NOP
RSEG      mydat2seg
              ; mydat2seg DATA segment selection
```

che assoluti (all'atto della stesura del programma si scrive direttamente l'indirizzo assoluto dove porre il codice o i dati in questione). L'uso di CSEG è comodo quando si sviluppano pezzi di programma che saranno poi utilizzati in altre applicazioni.

# L'Assembler 3

Infine ci sono le istruzioni vere e proprie che il compilatore tradurrà in linguaggio adatto al micro, inserendolo nella sua memoria, quando il programma sarà stato completamente verificato.

Si tratta di tutte quelle istruzioni che servono a comandare e gestire il flusso del programma del micro:

| | | | | | |
|---|---|---|---|---|---|
| ACALL | Absolute call | ADD | Add | ADDC | Add with carry |
| AJMP | Absolute jump | ANL | Logical and | CJNE | Compare & jump if not equal |
| CLR | Clear | CPL | Complement | DA | Decimal adjust |
| DEC | Decrement | DIV | Divide | DJNZ | Decrement & jump if not zero |
| INC | Increment | JB | Jump if bit set | JBC | Jump & clear bit if bit set |
| JC | Jump if carry set | JMP | Jump | JNB | Jump if bit not set |
| JNC | Jump if carry not set | JNZ | Jump if accum. not zero | JZ | Jump if accumulator zero |
| LCALL | Long call | LJMP | Long jump | MOV | Move |
| MOVC | Move code | MOVX | Move external | MUL | Multiply |
| NOP | No operation | ORL | Inclusive or | POP | Pop stack |
| PUSH | Push stack | RET | Return | RETI | Return from interrupt |
| RL | Rotate left | RLC | Rotate left thru carry | RR | Rotate right |
| RRC | Rotate right thru carry | SETB | Set bit | SJMP | Short jump |
| SUBB | Subtract with borrow | SWAP | Swap nibbles | XCH | Exchange bytes |
| XCHD | Exchange digits | XRL | Exclusive or | CALL | Generic call |

Table 2-3: 8051 Instructions and Mnemonics

E' inutile insistere sul fatto che la sintassi può cambiare da micro e micro, ma soprattutto tra IDE ed IDE. In modo che le società produttrici dell'IDE stesso si cautelano sugli altri sistemi di sviluppo.

# L'Assembler ed il C

E' possibile, anzi auspicabile, sviluppare progetti che richiedono sia potenzialità e semplicità del C con l'uso di certe funzioni elementari assembler.

Il C e l'assembler possono convivere in un IDE. Ci sono diversi modi in cui la convivenza è realizzata.

Uno è per esempio l'uso dell'istruzione asm in C, che consente di scrivere direttamente istruzioni assembler in zone di codice C.

Un'altra possibilità è quella di costruire un file in assembler. Questo file deve soddisfare a qualche semplice regola per potere essere usato dal programma principale in C. Un classico esempio si ha quando le variabili da passare hanno il nome preceduto dal simbolo: _

L'inverso è anche verificabile: in un programma assembler possiamo usare variabili definite in C ed anche funzioni definite in C. Solo che ancora la sintassi dipende da IDE ad IDE.